

```

*****
* Function: control_engine_follower
*
* Description:
*   This function sets the override_control_mode to no over ride so that
*   the engine follows the accelerator demand.
*
*****
```

```

static void control_engine_follower(void)
{
#define POSITIVE_PEDAL_TRANSITION_TIME    25 /* 250 MSEC */
#define NEGATIVE_PEDAL_TRANSITION_TIME    40 /* 400 MSEC */

engine_status = ENGINE_FOLLOWER_MODE;

if ((accelerator_pedal_position >= 5) && /* positive pedal transition */
    (accelerator_pedal_position_old <= 4) &&
    (low_speed_latch == FALSE))
{
    positive_pedal_trans = TRUE;
    zero_flywheel_trq_time = POSITIVE_PEDAL_TRANSITION_TIME;
    if (zero_flywheel_trq_timer >= NEGATIVE_PEDAL_TRANSITION_TIME)
        zero_flywheel_trq_timer = 0;
}

else
{
    if ((accelerator_pedal_position <= 4) && /* negative pedal transition */
        (accelerator_pedal_position_old >= 5) &&
        (low_speed_latch == FALSE))
    {
        zero_flywheel_trq_time = NEGATIVE_PEDAL_TRANSITION_TIME;
        zero_flywheel_trq_timer = 0;
    }
}

if ((zero_flywheel_trq_timer < zero_flywheel_trq_time) &&
    (current_gear > 1) && (current_gear < 10) && (low_speed_latch == FALSE))
{
    engine_control = TORQUE_CONTROL;
    command_ETC1 = C_ETC1_OVERSPEED;
    desired_engine_pct_trq = needed_percent_for_zero_flywheel_trq;

    if (actual_engine_pct_trq < (needed_percent_for_zero_flywheel_trq + 5));
        zero_flywheel_trq_timer++;
}

else
{
    if ((positive_pedal_trans == TRUE) && (low_speed_latch == FALSE))
    {
        positive_pedal_trans = FALSE;
        engine_commands = ENGINE_RECOVERY; /* engine: finish torque return */
        control_engine_recovery();
    }

    else
    {
        engine_control = OVERRIDE_DISABLED;
        command_ETC1 = C_ETC1_NORMAL;
    }
}

/* if predip_mode had been forced, this is the place to clear its flag. */
/* But only clear it if the pedal is depressed. This is for the case when */
/* a false confirmed gear is seen but the driver actually is coasting down. */
if (accelerator_pedal_position > 4)
    forced_predip = FALSE;
}
#endif
#endif
#endif

```

Best Available Copy

```
*****
* Function: control_engine_idle
*
* Description:
*   This function sets the engine controls for the idle mode.
*****
static void control_engine_idle(void)
{
    engine_control = OVERRIDE_DISABLED;
    command_ETC1 = C_ETC1_NORMAL;

    engine_status = ENGINE_IDLE_MODE;
}

#pragma EJECT
```

```
*****  
*  
* Function: control_engine_start  
*  
* Description:  
*   This function sets the engine controls for the start mode.  
*****/  
  
static void control_engine_start(void)  
{  
    engine_control = OVERRIDE_DISABLED;  
    command_ETC1 = C_ETC1_NORMAL;  
  
    engine_status = ENGINE_START_MODE;  
}  
  
#pragma EJECT
```

```
*****
* Function: control_engine_compression_brake
*
* Description:
*   This function controls the state of the engine compression brake.
*   The brake can be used during upshifts to speed up the decel rate of
*   the input shaft.
*****
static void control_engine_compression_brake(void)
{
    if (engine_communication_active &&
        (engine_status == ENGINE_SYNC_MODE) &&
        (shift_type == UPSHIFT) &&
        (input_speed_filtered > (gos + 150)) &&
        (destination_gear > 1) &&
        (destination_gear < 7) &&
        (engine_brake_available) &&
        ((dos_predicted < dos_prtd_lim_no_jake) || eng_brake_assist))
    {
        eng_brake_assist = TRUE;
    }
    else
    {
        eng_brake_assist = FALSE;
    }

    eng_brake_assist = FALSE; /* debug - force false state for now */
}

#pragma EJECT
```

```
*****
* Function: determine_gos
*
* Description:
*   This function multiplies the destination gear ratio times the
*   output shaft speed for use in the DRL_CMDS module.
*
*   gos = (g)ear * (o)utput (s)peed
*
*****/
```

```
static void determine_gos(void)
{
    /*** determine gos for the destination_gear ***/
    _bx = trn_tbl.gear_ratio[destination_gear + GR_OFS];
    _cx = output_speed_filtered;      /* output speed */
    asm mulu _cxdx, _bx;           /* BIN 8 result */
    asm shr1 _cxdx, #8;            /* make BIN 0 */
    gos = _cx;                      /* BIN 0 */

    _bx = trn_tbl.gear_ratio[destination_gear + GR_OFS];
    _cx = *(uint *)&dos_filtered;
    asm mul _cxdx, _bx;
    asm div _cxdx, #256;
    dgos = *(int *)&_cx;

    gos_signed = (signed int)(gos); /* allow signed math in other functions*/

    /*** determine gos for the "current_gear" ***/
    _bx = trn_tbl.gear_ratio[current_gear + GR_OFS];
    _cx = output_speed_filtered;      /* output speed */
    asm mulu _cxdx, _bx;           /* BIN 8 result */
    asm shr1 _cxdx, #8;            /* make BIN 0 */
    gos_current_gear = _cx;         /* BIN 0 */
}

#endif

#pragma EJECT
```

```

*****+
* Function: determine_shiftability_variables
*
* Description:
*   This function filters both the output speed and the rate of change of
*   the output speed for use in the shiftability function. This function
*   also calculates the rate of change of the input shaft based on the
*   filtered value for the rate of change of the output shaft.
*
*   The filters used in this function are required to get a high level of
*   stability. The BIN 8 filter used here will provide a much smoother
*   output which is needed to filter out driveline oscillations.
*
*   Note: These calculations were placed in this module because it is
*   called on a regular 10 msec interval. These calculations should
*   be placed in the pr_i_s_i.c96 module once shiftability is proven.
*   These variables are used in the SEL_GEAR.C96 module.
*
*****/

static void determine_shiftability_variables(void)
{
    /* LPF coefficients: exp(-WT), T=0.010s */
#define OS_LPF      248      /* 0.9691 BIN 8 (0.50Hz) */
#define DOSFK1      249      /* 0.9727 BIN 8 (0.44Hz) */
#define EPTFK1      252      /* 0.9844 BIN 8 (0.25Hz) */

#define IS_FK1       235      /* 0.9219 BIN 8 (0.??Hz) */
#define OS_FK1       236      /* 0.9219 BIN 8 (0.??Hz) */
#define DISFK1       236      /* 0.9219 BIN 8 (0.??Hz) */

#define LOW_RANGE    3197     /* 3.1224 BIN 10          */
#define BIN_10       .1024

    static long dos_filtered_bin8;
    static int ept_filtered_bin8;

    unsigned long is_filtered_partial_1;
    unsigned long is_filtered_partial_2;
    unsigned long os_filtered_partial_1;
    unsigned long os_filtered_partial_2;

    /** create lpf_output_accel **/
    _bx = *(uint *)&output_speed_accel;
    _cx = *(uint *)&lpf_output_accel - _bx;
    asm mul _cxdx, #OS_LPF;
    asm div _cxdx, #256;
    _bx += _cx;
    lpf_output_accel = *(int *)&_bx;
    /* _bx = x(n), BIN 0 */
    /* _cx = y(n-1) - x(n), BIN 0 */
    /* _cxdx = K*(...), BIN 8 */
    /* make BIN 0 */
    /* _bx = x(n) + K*(...), BIN 0 */
    /* save acceleration */

    /** dos_filtered = (dos_filtered * DOSFK1) + (lpf_output_accel * (1-DOSFK1)) **/
    _cxdx = *(ulong *)&dos_filtered_bin8;
    asm shrl _cxdx, #2;
    asm mul _cxdx, #DOSFK1;
    asm shrl _cxdx, #6;
    dos_filtered_bin8 = *(long *)&_cxdx;
    /* BIN 8 */
    /* BIN 6 (_cx) */
    /* BIN 14 */
    /* BIN 8 */
    /* save partial result */

    _cx = *(uint *)&lpf_output_accel;
    _bx = 256 - DOSFK1;
    asm mul _cxdx, _bx;
    dos_filtered_bin8 += *(long *)&_cxdx;
    /* BIN 0 */
    /* 1 BIN 8 - DOSFK1 */
    /* BIN 8 */
    /* sum is final result */

    dos_filtered = (int)(dos_filtered_bin8 >> 8); /* BIN 0 */

    /** eng_percent_torque_filtered = (eng_percent_torque_filtered * EPTFK1) +
        (net_engine_pct_trq * (1-EPTFK1)) **/
    _cx = *(uint *)&ept_filtered_bin8;
    asm mul _cxdx, #EPTFK1;
    asm shrl _cxdx, #8;
    ept_filtered_bin8 = *(int *)&_cx;
    /* BIN 8 */
    /* BIN 16 */
    /* BIN 8 */
    /* save partial result */

    _cx = net_engine_pct_trq;
    _bx = 256 - EPTFK1;
    asm mul _cxdx, _bx;
    ept_filtered_bin8 += *(int *)&_cx;
    /* BIN 0 */
    /* 1 BIN 8 - EPTFK1 */
    /* BIN 8 */
    /* sum is final result */
}

```

```

ang_percent_torque_filtered = (char)(opt_filtered_bin8 >> 8);

/** input_shaft_accel_calculated = dos_filtered * gear_ratio **/


_cx = trn_tbl.gear_ratio[destination_gear + GR_OFS]; /* BIN 8 */
_bx = *(uint *)&dos_filtered; /* BIN 0 */
asm mul _cxdx, _bx; /* BIN 8 */
asm shr1l _cxdx, #8; /* BIN 0 */
input_shaft_accel_calculated = *(int *)&_cx;

/** calculate filtered input and output shaft speeds for AutoSplit ***

/** determine os_based_on_rcs variable ***
if (output_speed < 1000)
{
    _bx = aux_speed; /* BIN 0 */
    _cx = BIN_10; /* BIN 10 */
    _ax = LOW_RANGE; /* BIN 10 */
    asm mulu _cxdx, _bx; /* make aux_speed BIN 10 */
    asm divu _cxdx, _ax; /* divide by low range BIN 10 */
    os_based_on_rcs = _cx;
}

/** input_speed_filtered = (input_speed_filtered * IS_FK1) +
    (input_speed * (1-IS_FK1)) **/

_ax = (is_filtered_bin8 >> 4); /* BIN 4 */
(cx = IS_FK1; /* BIN 8 */
asm mulu _axbx, _cx; /* BIN 12 */
asm shr1l _axbx, #4; /* BIN 8 */
is_filtered_partial_1 = _axbx; /* BIN 8 */

_cx = input_speed; /* BIN 0 */
_ax = 256 - IS_FK1; /* 1 BIN 8 - IS_FK1 */
asm mulu _axbx, _cx; /* BIN 8 */
is_filtered_partial_2 = _axbx; /* BIN 8 */

is_filtered_bin8 = is_filtered_partial_1 + is_filtered_partial_2;
input_speed_filtered = (unsigned int)(is_filtered_bin8 >> 8); /* BIN 0 */

/** output_speed_filtered = (output_speed_filtered * OS_FK1) +
    (output_speed * (1-OS_FK1)) **/

_ax = (os_filtered_bin8 >> 4); /* BIN 4 */
(cx = OS_FK1; /* BIN 8 */
asm mulu _axbx, _cx; /* BIN 12 */
asm shr1l _axbx, #4; /* BIN 8 */
os_filtered_partial_1 = _axbx; /* BIN 8 */

if (0)
if (output_speed < 250)
    _cx = os_based_on_rcs; /* BIN 0 */
else
endif
_cx = output_speed; /* BIN 0 */
_ax = 256 - OS_FK1; /* 1 BIN 8 - OS_FK1 */
asm mulu _axbx, _cx; /* BIN 8 */
os_filtered_partial_2 = _axbx; /* BIN 8 */

os_filtered_bin8 = os_filtered_partial_1 + os_filtered_partial_2;
output_speed_filtered = (unsigned int)(os_filtered_bin8 >> 8); /* BIN 0 */

/** input_speed_accel_filtered = (input_speed_accel_filtered * DISFK1) + (input_shaft_accel * (1-DISFK1)) **/


_cxdx = *(ulong *)&dis_filtered_bin8; /* BIN 8 */
asm shr1l _cxdx, #4; /* BIN 4 (_cx) */
asm mul _cxdx, #DISFK1; /* BIN 12 */
asm shr1l _cxdx, #4; /* BIN 8 */
dis_filtered_bin8 = *(long *)&_cxdx; /* save partial result */

_cx = *(uint *)&input_speed_accel; /* BIN 0 */
_bx = 256 - DISFK1; /* 1 BIN 8 - DISFK1 */
asm mul _cxdx, _bx; /* BIN 8 */
asm shr1l _cxdx, #4; /* save partial result */

```

```

input_speed_accel_filtered = (int)(dis_filtered_bnd >> 8); /* BIN 0 */

/** determine state of clutch **/ /*!!!!!! this is temporary until we get */
/*!!!! clutch state over J1939. !!!*/
#if (0)
    if (engine_speed > input_speed)
        clutch_slip_speed = engine_speed - input_speed;
    else
        clutch_slip_speed = input_speed - engine_speed;

    if (clutch_slip_speed > 200)
        clutch_state = DISENGAGED;
    else
        if ((engine_speed > 700) && (low_speed_latch == FALSE))
            clutch_state = ENGAGED; /* Note: 700 should be idle+100RPM */
#endif
/** Below is a known undesirable condition that could be corrected with the J1939
clutch state information. When input speed is brought below 700 RPM
while in gear and the clutch is disengaged to achieve gear box neutral this
algorithm holds the system in the follower mode until the driver bring the
engine speed above the 700 RPM limit. The 700 RPM limit is needed to prevent
false ENGAGED indications at idle speeds. **/
```

/* determine desired percent torque needed for zero torque at flywheel */

```

#if (0)
    /* This does not work but was not a problem to hard code a value
       because the accessory loading in this vehicle does not change
       much. This algorithm DOES need to work for the product. */
    if ((accelerator_pedal_position < 2) &&
        (clutch_state == ENGAGED) &&
        (current_gear == 0) &&
        (input_speed_filtered < 1100) &&
        ((engine_control == OVERRIDE_DISABLED) &&
         (low_speed_latch == FALSE) && (current_gear == 0)) ||
        (output_speed_filtered < 20)))
        percent_torque_accessories = eng_percent_torque_filtered; /* get at idle */
#endif
    percent_torque_accessories = 3; /* force value for now */

needed_percent_for_zero_flywheel_trq = percent_torque_accessories + ←
nominal_friction_pct_trq;
```

/* determine overall error across the transmission */

```

overall_error = ((signed int)(input_speed_filtered) - (signed int)(gos));

```

>

```

#pragma EJECT

```

```
*****
* Function: communicate_with_driveline
*
* Description:
*   This is the periodic task which controls the actions of the engine
*   by defining mode of control and controlling speed and torque output
*   levels depending upon the control function being performed. This task
*   is also intended for control of other driveline components (not yet
*   named) which may be available in the future.
*****

```

```
void communicate_with_driveline(void)
{
    initialize_driveline_data();

    x_start_periodic();
    while (1)
    {
        control_engine_compression_brake();

        determine_gos(); /* calculate (G)ear times the (O)utput (S)haft */

        determine_shiftability_variables();

        if ((engine_communication_active) &&
            (R747_type != BASE) &&
            (R747_type != DUAL_FORCE))
        {

            if ((desired_sync_test_mode == TRUE) && (output_speed_filtered < 100))
                control_engine_sync_test_mode();

            else
                /* start of normal engine_commands switch */

                switch (engine_commands)
                {
                    case ENGINE_PREDIP:
                        control_engine_predip();
                        break;

                    case ENGINE_SYNC:
                        control_engine_sync();
                        break;

                    case ENGINE_RECOVERY:
                        control_engine_recovery();
                        break;

                    case ENGINE_IDLE:
                        control_engine_idle();
                        break;

                    case ENGINE_START:
                        control_engine_start();
                        break;

                    case ENGINE_FOLLOWER:
                    default:
                        control_engine_follower();
                        break;
                }
            /* end of normal engine_commands switch */

            switch (eng_brake_command)
            {
                case ENG_BRAKE_OFF:
                    retarder_control = TORQUE_CONTROL;
                    desired_retarder_pct_trq = 0;
                    break;

                case ENG_BRAKE_FULL:
                    retarder_control = TORQUE_CONTROL;
                    desired_retarder_pct_trq = -100;
                    break;

                case ENG_BRAKE_IDLE:

```

```
default:  
    retarder_control = OVERRIDE_DISABLED;  
    desired_retarder_pct_trq = 0;  
    break;  
}  
}  
else  
    engine_status = ENGINE_NOT_PRESENT;  
  
/* store old value for use in "control_engine_follower" function */  
accelerator_pedal_position_old = accelerator_pedal_position;  
  
x_sync_periodic(US_PER_LOOP);  
}  
x_end_periodic();  
}
```

```
*****
* Unpublished and confidential. Not to be reproduced,
* disseminated, transferred or used without the prior
* written consent of Eaton Corporation.
*
* Copyright Eaton Corporation, 1994
* All rights reserved.
*****
* Filename: pr_s_i_s.c96 (R-747) (AutoSplit)
*
* Description:
*   The modules contained within this compilation unit are
*   intended to implement functionality of the Process System
*   Input Signals task defined in the design documentation.
*   In general, Analog to digital conversions are started on
*   Port0. The necessary hardware initialization and variable
*   initialization for inputs on Port0 are handled. The switch
*   inputs are captured to avoid conflict with AD conversions
*   and all necessary scaling and error check for these inputs
*   is conducted.
*
* Part Number: <none>
*
* $Log: ?
*
*   Rev 1.3 9 Dec 1994 15:06: markyvech
*   Added "trns_act.h" to the includes so that R747_type could be use to
*   determine if the electric shift knob is a splitter type or a intent to
*   shift type.
*
*   Rev 1.2 6 Dec 1994 15:06: markyvech
*   Converted for use with R-747 program. (Added clutch & range switches)
*   Also re-scaled ECU-B ignition A2D code to work in ECU-II.
*
*   Rev 1.1 19 May 1994 11:32:26 markyvech
*   Converted for use with AutoSplit ECU2
*
*   Rev 1.0 12 Sep 1991 08:04:26 amsallen
*   Initial revision.
*****
*/
```

```
*****
* Header files included.
*
*****
```

```
#include <exec.h>          /* executive information      */
#include <kr_sfr.h>          /* KR special function registers */
#include <kr_def.h>          /* KR definitions            */
#include <c_regs.h>          /* KR internal register definitions */
#include <wwslib.h>          /* world wide software definitions */
#include "pr_s_i_s.h"         /* process system input signal information */
#include "sysgen.h"           /* defines the task names and priority */
#include "cont_sys.h"
#include "trns_act.h"

*****
*
* #defines local to this file.
*
*****
```

```
/* Start_AD_Conversions */

#define ENABLE_AD PTS_SCAN 0X20
#define ENABLE_AD_ISR 0X20

#define PERIOD 10U                  /* 10ms */
#define RKM_PERIOD 50U               /* 50ms */
*****
```

```

* Constants and variables declared by this file.
.

*****  

/* Digital Inputs on Port1 */

uchar splitter_select_switch;
uchar intent_to_shift_switch;
uchar intent_hold;
uchar intent_hold_timer;
uchar range_select_switch;
uchar in_gear_switch;
uchar splitter_launch_state;
uchar clutch_state;

/* Analog Inputs on Port0 */

int ignition_volts;
int splitter_position;

#define IGNITION_VOLTS_CHANNEL_RESULT    1
#define SPLITTER_POS_CHANNEL_RESULT     15

#define CONVERSION_TIME 0xef          /* for state time = 125 nsec:      */
/*           sample time = 3.6250 usec   */
/*           convert time = 20.1875 usec */
#define CONVERT_8 8                  /* scan and convert 8 channels   */

#define CONVERT_IGNITION_VOLTS         (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_0)
#define UNUSED_CHANNEL_1              (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_1)
#define UNUSED_CHANNEL_2              (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_2)
#define UNUSED_CHANNEL_3              (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_3)
#define UNUSED_CHANNEL_4              (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_4)
#define UNUSED_CHANNEL_5              (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_5)
#define UNUSED_CHANNEL_6              (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_6)
#define CONVERT_SPLITTER_POS          (_NORM_MODE|_10_BIT_MODE|_STRT_CONV|_CHNL_7)
#define STOP_CONVERSION               0x00
#define START_CONVERSION              _ad_command = CONVERT_IGNITION_VOLTS

/* table containing AD_result and AD_Command values after PTS scan */
unsigned int AD_Table[16];

/* AD SCAN PTS CONTROL BLOCK LOCATION */
_ad_ptsb_type AD_Con_Block;
#pragma locate(AD_Con_Block=0x01F8)      /* locate pts control block */
#pragma pts(AD_Con_Block = 5)            /* set pts vector 5, A/D done */

#pragma EJECT

```

```

*****
* Function: Initialize_Input_Signals
*
* Description:
*   This routine initializes the A/D converter. It sets the A/D to
*   run in PTS scan mode, 10bit conversion. The PTS control block is
*   set up and the Command/result table is initialized.
*
*****
```

```

void Initialize_Input_Signals(void)
{
    /* if we knew when the first speed packet arrived, we could initialize
       with those values. since we don't, be safe and use zero. */

    AD_Table[0] = UNUSED_CHANNEL_1;           /* place holder for channel 1 */
    AD_Table[1] = 0x0000;                     /* IGNITION_VOLTS_CHANNEL_RESULT */
    AD_Table[2] = UNUSED_CHANNEL_2;           /* place holder for channel 2 */
    AD_Table[3] = 0x0000;                     /* UNUSED_1_RESULT */
    AD_Table[4] = UNUSED_CHANNEL_3;           /* place holder for channel 3 */
    AD_Table[5] = 0x0000;                     /* UNUSED_2_RESULT */
    AD_Table[6] = UNUSED_CHANNEL_4;           /* place holder for channel 4 */
    AD_Table[7] = 0x0000;                     /* UNUSED_3_RESULT */
    AD_Table[8] = UNUSED_CHANNEL_5;           /* place holder for channel 5 */
    AD_Table[9] = 0x0000;                     /* UNUSED_4_RESULT */
    AD_Table[10] = UNUSED_CHANNEL_6;          /* place holder for channel 6 */
    AD_Table[11] = 0x0000;                     /* UNUSED_5_RESULT */
    AD_Table[12] = CONVERT_SPLITTER_POS;      /* Command convert splitter pos */
    AD_Table[13] = 0x0000;                     /* UNUSED_6_RESULT */
    AD_Table[14] = STOP_CONVERSION;          /* command to Stop conversions */
    AD_Table[15] = 0x0000;                     /* SPLITTER_POS_CHANNEL_RESULT */

    AD_Con_Block.cnt = CONVERT_8;
    AD_Con_Block.ctrl = _AD_MODE|_S_D_UPDT;    /* A/D mode bits 0,1 of PTS_CONTROL */
                                                /* always set to 3h bit 2 = 0 */
                                                /* S/D update at end of cycle */
                                                /* bit 5 always 0 */
                                                /* Set mode for AD SCAN */

    AD_Con_Block.s_d = AD_Table;               /* Load s_d with AD_Table address */
    AD_Con_Block.reg = (void *)&_ad_result;    /* Load reg with AD_Result address */

    _ad_time = CONVERSION_TIME;
    _ad_test = _NO_OFFSET;                    /* Disable test mode */
    _pts_select &= ~(_PTS_ADDONE_BIT);        /* Disable AD PTS */
}

#endif

```

```
#pragma EJECT
```

```
*****
*
* Function: AD_ISR
*
* Description:
*   This interrupt service routine resets the PTS COUNT, pts_sd and pts_reg
*   for another PTS_Scan A/D cycle. It also readies a task to run when
*   a PTS cycle has completed.
*
*****
```

```
#pragma interrupt(AD_ISR=5)
void AD_ISR(void)
{
    x_start_isr();
    AD_Con_Block.cnt = CONVERT_8;           /* Reset pts count for next cycle */
    AD_Con_Block.s_d = AD_Table;           /* Reset table pointer to start of table */
    AD_Con_Block.reg = (void *)&_ad_result;

    x_ready(PREPROCESS_INPUT_SIGNALS);      /* Ready pr_s_i_s task */
    x_end_isr();
}
```

```
#pragma EJECT
```

```
*****
* Function: Start_AD_Conversions
*
* Description:
*   This function initializes the input signal processing function
*   if it has not already been done, and then starts the PTS Scan
*   of the AD channels by sending the appropriate command to the
*   ad_command register.
*
*****
```

```
void Start_AD_Conversions(void)
{
    if ((_int_mask & ENABLE_AD_ISR) == 0 )
        Initialize_Input_Signals();           /* Set up AD table for PTS */

    _pts_select |= ENABLE_AD PTS_SCAN;
    _int_mask |= ENABLE_AD_ISR;

    x_preamr_stimulus();

    START_CONVERSIONS; /* Start a conversion, initiate the PTS cycles */
    x_wait_stimulus(); /* AD_ISR will ready task when PTS is complete */
}

#pragma EJECT
```

```
*****
* Function: read_switch_inputs
*
* Description:
*   Read the state of the digital inputs.
*
*****
```

```
void read_switch_inputs(void)
{
    if (port_1_switches & 0x1)      /* P1.0 */
        in_gear_switch = TRUE;
    else
        in_gear_switch = FALSE;

    if (port_1_switches & 0x2)      /* P1.1 */
        range_select_switch = LOW;
    else
        range_select_switch = HIGH;

    if (R747_type == INTENT)
    {
        if (port_1_switches & 0x4)      /* P1.2 */
        {
            intent_to_shift_switch = FALSE;

            if (intent_hold_timer < 25)
                intent_hold_timer += 1;
            else
                intent_hold = FALSE;
        }
        else
        {
            intent_to_shift_switch = TRUE;
            intent_hold_timer = 0;
        }
    }

    else
    {
        if (port_1_switches & 0x4)      /* P1.2 */
            splitter_select_switch = HIGH;
        else
            splitter_select_switch = LOW;
    }

    if ((port_2_switches & 0x30) == (0x10))
        clutch_state = ENGAGED;
    else
        clutch_state = DISENGAGED;

#if (0)
    if (port_1_switches & 0x4)
        splitter_launch_state = LO_SPLITTER;
    else
        splitter_launch_state = HI_SPLITTER;
#endif

    splitter_launch_state = LO_SPLITTER;
}

#pragma EJECT
```

```
*****
* Function: process_input_signals
*
* Description:
*   This is the periodic task which starts the analog conversions on Port0.
*****
void process_input_signals(void)
{
    intent_hold = FALSE; /* initialize intent_hold */

    x_start_periodic();
    while (1)
    {
        read_switch_inputs();

        Start_AD_Conversions();

        /* Clean up and scale AD values */
        ignition_volts = scale_system_ad_inputs(IGNITION_VOLTAGE);
        splitter_position = scale_system_ad_inputs(SPLITTER_POSITION);

        x_sync_periodic(PERIOD*1000);
        /* x_sync_periodic(RKM_PERIOD*1000); */
    }
    x_end_periodic();
}

#pragma EJECT
```

```

*****+
* Function: scale_system_ad_inputs
*
* Description:
*   This function removes the channel, status and reserved bits from
*   the raw AD values, and performs all necessary scaling and error
*   checking for the analog inputs on Port0.
*
*****+

int scale_system_ad_inputs(char Channel)
{
    int Scaled_Value = 0;
    uint volts_per_bit; /* BIN 16 */
    uint units_per_bit; /* BIN 16 */
    /* #define TWELVE_VOLT_FULL_SCALE      22.46 /* ECU_8 volts */
    #define TWELVE_VOLT_FULL_SCALE      34.51 /* ECU_2 volts */
    #define TWENTY_FOUR_VOLT_FULL_SCALE 40.49 /* volts */
    #define DISTANCE_FULL_SCALE        100 /* */

    volts_per_bit = (uint)((TWELVE_VOLT_FULL_SCALE*65536/1023)+0.5);
    units_per_bit = (uint)((DISTANCE_FULL_SCALE*65536/1023)+0.5);

    switch (Channel)
    {
    case 0: /* IGNITION VOLTAGE */
        _cx = AD_Table[IGNITION_VOLTS_CHANNEL_RESULT] >> 6;
        asm mulu _cxdx, volts_per_bit; /* volts, BIN 16 (_dx, BIN 0) */
        Scaled_Value = *(int *)&_dx;
        break;

    case 1: /* UNUSED */ /* to be completed when a product requires it */
        Scaled_Value = (0);
        break;

    case 2: /* UNUSED */ /* to be completed when a product requires it */
        Scaled_Value = (0);
        break;

    case 3: /* UNUSED */ /* to be completed when a product requires it */
        Scaled_Value = (0);
        break;

    case 4: /* UNUSED */ /* to be completed when a product requires it */
        Scaled_Value = (0);
        break;

    case 5: /* UNUSED */ /* to be completed when a product requires it */
        Scaled_Value = (0);
        break;

    case 6: /* UNUSED */ /* to be completed when a product requires it */
        Scaled_Value = (0);
        break;

    case 7: /* SPLITTER POSITION */
        _cx = AD_Table[SPLITTER_POS_CHANNEL_RESULT] >> 6;
        asm mulu _cxdx, units_per_bit; /* distance, BIN 16 (_dx, BIN 0) */
        Scaled_Value = *(int *)&_dx;
        break;

    default:
        break;
    }
    return (Scaled_Value);
}

```

```
*****
*          Unpublished and confidential. Not to be reproduced,
*          disseminated, transferred or used without the prior
*          written consent of Eaton Corporation.
*
*          Copyright Eaton Corporation, 1991-94.
*          All rights reserved.
*****
* Filename: seq_shft.c96      (R-747) (AutoSplit)
*
* Description:
*   The functions in this file will perform the required system
*   level operations for implementing Sequence Shift.
*
* Part Number: <none>
*
*   Rev 1.2  12 Dec 1994 09:15      markyvech
* In function "sequence_shift()" added code for the intent_hold.
*
*   Rev 1.1  09 Dec 1994 08:07      markyvech
* In function "sequence_shift()" added code for the intent to shift
* feature. This will allow shift_initiate, (predip mode), to be called.
* Also added code to allow for the cancellation of intent to shift if
* conditions allow, (call confirm shift).
*
*   Rev 1.0  12 May 1994 16:26:00  markyvech
* Initial version
*****
/* Header files included.
*/
#include <exec.h>           /* executive information */
#include <c_regs.h>          /* KR internal registers */
#include <wwslib.h>           /* World Wide Software Library */
#include "cont_sys.h"         /* control system information */
#include "conj1939.h"          /* Defines interface to engine communications info */
#include "con_o_s.h"          /* control output signal information */
#include "drv_cmds.h"          /* driveline commands information */
#include "sel_gear.h"
#include "shf_tbl.h"           /* Contains information relative to engine */
#include "trn_tbl.h"           /* transmission table information */
#include "trns_act.h"          /* transmission information */
#include "pr_s_i_s.h"          /* process system input signals information */

/*
* #defines local to this file.
*/
/*
* publics.
*/
unsigned char forced_predip_timer;
unsigned char init_dest_gear_timer;
signed char initial_destination_gear;

/*
* Constants and variables declared by this file.
*/
static uchar coast_mode;      /* allows vehicle to coast in low gears */

#define FORCED_PREDIP_TIME    150    /* 300 MSEC at 5 counts per loop */
#define IN_SYNC_MODE_TIME_LIMIT 200    /* 2 SEC */
```

~~Program EJECT~~

```
*****  
*  
* Functions: initialize_sequence_shift  
*  
* Description:  
*   This function initializes those module variables that must be set to a  
*   known state on power up or reset.  
*  
*****/  
  
void initialize_sequence_shift(void)  
{  
    shift_type = UPSHIFT;  
    shift_in_process = FALSE;  
    forced_preset_timer = 0;  
}  
  
#pragma EJECT
```

```
*****
* Function: shift_initiate
*
* Description:
*   This function begins the shift sequence by setting up the
*   transmission to pull to Neutral, commands the electronic engine
*   controller to go to zero torque and prepares the clutch to disengage
*   if required.
*
*****/
```

```
static void shift_initiate(void)
{
    /* (do not request engine fueling with engine brake on) */
    eng_brake_command = ENG_BRAKE_OFF;      /* eng brake: zero torque */

    if ((lpf_output_speed < shf_tbl.min_output_spd) ||
        (clutch_state == DISENGAGED) ||
        ((coast_mode) && (shift_type != UPSHIFT) &&      /* Prevents engine control */
        ((destination_gear < 3) ||                      /* in low gear(s) downshifts. */
        ((destination_gear < 4) && (accelerator_pedal_position <= 5)))))

    {
        engine_commands = ENGINE_FOLLOWER;
    }
    else
    {
        engine_commands = ENGINE_PREDIP;      /* engine: bring torque to zero */
        coast_mode = FALSE;
    }
}

#pragma EJECT
```

```

*****+
* Function: synchronize_gear
*
* Description:
*   This function assists the synchronizing of the transmission by
*   utilizing SAE J1939 functions to control an electronic engine.
*
*****/

static void synchronize_gear(void)
{
    /* turn on engine brakes (J1939) if engine brake assisted shift is requested */

    if (eng_brake_assist)
        eng_brake_command = ENG_BRAKE_FULL;
    else
        eng_brake_command = ENG_BRAKE_OFF;

    if ((lpf_output_speed < shf_tbl.min_output_spd) ||
        (clutch_state == DISENGAGED) ||
        ((coast_mode) && (shift_type != UPSHIFT) &&      /* Prevents engine control */
        ((destination_gear < 3) ||                         /* in low gear(s) downshifts. */
        ((destination_gear < 4) && (accelerator_pedal_position <= 5))))
    {
        engine_commands = ENGINE_FOLLOWER;
    }
    else
    {
        engine_commands = ENGINE_SYNC;
        coast_mode = FALSE;
    }

    if ((engine_status != ENGINE_SYNC_MODE) &&
        (engine_commands == ENGINE_SYNC))
        init_dest_gear_timer = 0;

    if (init_dest_gear_timer < 8)                                /* Save the initial gear */
    {                                                        /* to check for a new one */
        initial_destination_gear = destination_gear_selected; /* as the shift progresses. */
        init_dest_gear_timer++;
    }
    else
        if ((engine_status == ENGINE_SYNC_MODE) &&
            (initial_destination_gear != destination_gear_selected)) /* If the gear changes, */
            /* force the predip mode */
            /* to allow the splitter */
            /* to move. */
        {
            forced_predip_timer = FORCED_PREDIP_TIME;
            forced_predip = TRUE;
        }
    }

#endif

#pragma EJECT

```

```
*****
* Function: confirm_shift
*
* Description:
*   This function finishes the shift; shift_in_process is set FALSE.
*****
static void confirm_shift(void)
{
    engine_commands = ENGINE_RECOVERY;      /* engine: finish torque return */
    eng_brake_command = ENG_BRAKE_OFF;       /* eng brake: zero torque */
    shift_in_process = FALSE;
    coast_mode = TRUE;
}

#pragma EJECT
```

```

*****+
* Function: sequence_shift
*
* Description:
*   This function calls the appropriate procedures to perform the
*   operations of Sequence_Shift depending on the current state of
*   the shift process.
*
*****+
void sequence_shift(void)
{
    if (destination_gear < last_known_gear) /* determine shift type */
    {
        if (pct_demand_at_cur_sp > 5)
            shift_type = POWER_DOWN_SHIFT;
        else
            shift_type = COAST_DOWN_SHIFT;
    }
    else
        shift_type = UPSHIFT;

    if ((forced_predip_timer >= 4) && /* Time out a forced return to the predip mode */
        (g_ptr == 0)) /* if the destination gear selected changes */
        forced_predip_timer -= 4; /* during the sync mode. */

    if (forced_predip_timer > 0)
        forced_predip_timer--;

    if (destination_gear == NULL_GEAR) /* System has reset: do not start a shift */
    {
        engine_commands = ENGINE_FOLLOWER;
        eng_brake_command = ENG_BRAKE_IDLE;
    }

    else
        if (((transmission_position == OUT_OF_GEAR) &&
             (forced_predip_timer == 0) &&
             ((engine_status == ENGINE_SYNC_MODE) ||
              (engine_status == ENGINE_PREDIP_MODE))) /* forces shift_initiate() */
        {
            synchronize_gear();
        }

        else
            if (((((engine_status == ENGINE_SYNC_MODE) ||
                  (engine_status == ENGINE_RECOVERY_MODE)) &&
                  (forced_predip_timer == 0) &&
                  (destination_gear == current_gear) &&
                  (transmission_position == IN_GEAR)) ||
                  ((shift_in_process == TRUE) && /* cancel intent to shift if */
                   (intent_to_shift_switch == FALSE) && /* conditions allow it. */
                   (shift_init_type == MANUAL) &&
                   (automatic_sip == 0) &&
                   (transmission_position == IN_GEAR) &&
                   (engine_status == ENGINE_PREDIP_MODE)))
            {
                confirm_shift();
            }

            else
                if (((destination_gear != current_gear) && /* auto splitter */
                     (low_speed_latch == FALSE) &&
                     (automatic_sip != 0) &&
                     (transmission_position == IN_GEAR)) ||
                     ((intent_to_shift_switch == TRUE) && /* Allows for intent_to_shift */
                      (desired_gear != destination_gear_selected) && /* manual shift. */
                      (intent_hold == FALSE) &&
                      (automatic_sip == 0) &&
                      (shift_init_type == MANUAL) &&
                      (low_speed_latch == FALSE) &&
                      (transmission_position == IN_GEAR)) ||
                     (forced_predip_timer > 0) || /* gear changed during shift */

```

```
((transmission_position == OUT_OF_GEAR) && /* manual shift */  
 (low_speed_latch == FALSE)))  
{  
   shift_initiate();  
 }  
)
```

```
*****
*          Unpublished and confidential. Not to be reproduced,
*          disseminated, transferred or used without the prior
*          written consent of Eaton Corporation.
*
*          Copyright Eaton Corporation, 1993-94.
*          All rights reserved.
*****
* Filename: sel_gear.c96      (R-747) (AutoSplit)
*
* Description:
*   This module is the periodic task "select_gear". It assigns values
*   to destination_gear_selected as a function of selected_mode, input
*   and output shaft speeds.
*   Shift parameters are in the data structure shf_tbl.
*
* Part Number: <none>
*
*   Rev 1.1  12 Dec 1994 08:05      markyvech
* In function "determine_manual_shift_pts()", changed auto_up_rpm from
* 1350 RPM to 1425 RPM because the transmission has been changed to a "8"
* ratio and skip upshifts need to be avoided.
*
*   Rev 1.0  14 Sep 1994 14:02:00      markyvech
* Initial revision.
*****
/*****
* Header files included.
*****
#include <exec.h>          /* executive information */
#include <c_regs.h>          /* c registers */
#include <wwslib.h>          /* contains common global defines */
#include "cont_sys.h"        /* control system */
#include "conj1939.h"         /* defines interface to j1939 control module */
#include "drl_cmds.h"         /* driveline commands information */
#include "sel_gear.h"          /* select gear */
#include "shf_tbl.h"          /* shift table definition */
#include "trn_tbl.h"          /* (system) transmission table definition */
#include "calc_spd.h"
#include "trns_act.h"
#pragma noreentrant
*****
* #defines local to this file.
*****
#define US_PER_LOOP 40000U
#define INITIAL_START_GEAR 1
#define SHIFT_INHIBIT_DECEL_LIMIT -150
*****
* Constants and variables declared by this file.
*****
/* public */

char destination_gear_selected;
char destination_gear;
char flash_desired_allowed;
char desired_gear;
char desired_gear_dn;
char desired_gear_up;
uchar coasting_latch;
uchar shift_init_type;
uint lpf_output_speed;
```

```

int dos_predicted;           /* BIN 0 */
int dos_prtd_lim_no_jake;   /* BIN 0 */

/* local */

/* filter weights for the "mda" output speed filter */
static register uchar w1;
static register uchar w2;
static register uchar w3;
static register uchar w4;

/* shift table (define and extern in shf_tbl.h) */
struct shf_tbl_t shf_tbl;

/* default shift table values */
const struct shf_tbl_t ini_shf_tbl =
{
    1200,          /* aut_dwn_rpm */
    1000,          /* aut_min_dwn_rpm */
    1700,          /* aut_up_rpm */
    0,             /* best_gr_offset */
    50,            /* dwn_offset_rpm */
    100,           /* dwn_reset_rpm */
    400,           /* dwn_timer_offset_rpm */
    40,            /* hysteresis_rpm */
    1850,          /* man_dwn_sync_rpm */
    700,           /* man_up_sync_rpm */
    1900,          /* rated_rpm */
    150,           /* up_offset_rpm */
    125,           /* up_reset_rpm */
    200,           /* up_timer_offset_rpm */
    0,              /* dwn_accel */
    8,              /* up_accel */
    3000,          /* offset_time */
    (uint)(0.25*256), /* aut_up_pct */
    10,             /* min_output_spd */
    1,              /* max_start_gear */
    0,              /* padbyte1 */

    196,           /* k1_ability, min-ft/rev-sec, BIN 12 */
    431,           /* axle_ratio_cal, BIN 7 */
    383,           /* gcw_k1, rev/sec-min-ft, BIN 0 */
    2437,          /* gcw_k2, rev/sec^2, BIN 7 */
    1325,          /* calc_start_point, rpm, BIN 0 */

    107,           /* k6_ability, min-lb-ft-sec/rev, BIN 8 */
    1500,          /* auto_up_lo_base, rpm, BIN 0 */
    1100,          /* auto_dn_lo_base, rpm, BIN 0 */
    100,           /* auto_rtd_offset, rpm, BIN 0 */
    1000,          /* lowest_engage_rpm, BIN 0 */
    0,              /* padword1 */
    0,              /* padword2 */
};

/* local -- initialized at start of task by select_gear */

/* shift points with anti-hunt offsets; referenced by auto_downshift and
   auto_upshift; set by get_automatic_gear and select_gear */
uint upshift_point;
uint downshift_point;

/* lower limit for gear selections */
char lowest_forward;

/* indicate direction of a get_automatic_gear shift; referenced by
   get_manual_gear; cleared by select_gear when shift complete */
char automatic_sip;

/* used in the determination of shift_points based on throttle position */
static uint auto_up_rpm;
static uint auto_dn_rpm;
static uint auto_up_offset_rpm;
static uint auto_dn_offset_rpm;

/* delay counter for anti-hunt */
static uchar antihunt_counter;

/* gross combined weight calculations */

```

```

#define ZERO_SPEED_TIME_LIMIT 4500 /* 3 min a 0.040 period */
#define VALID_OLD_DATA_TIME 100 /* 4 sec a 0.040 period */
#define MAX_VEHICLE_WEIGHT 100000 /* 100,000 lbs */
#define DOS_OFFSET_INIT 48
#define ENG_DECEL_LOW_LIMIT -350 /* rpm/s */
#define ENG_DECEL_LPF 226 /* exp(-2pi(0.53Hz)(0.040s)) = 0.875 (BIN 8) */

#if (0)
static ulong gross_combined_weight;
static long gcw_local;
static uint gcw_local_counter;
static ulong gcw_local_total;
static uchar valid_old_data_counter;
static uchar missed_shift_offset_arm;
static uchar gcw_calculation_allowed;
static uint zero_speed_counter;
static uchar gcw_first_shift;
static uchar dos_filter_latch;
static uint gcw_cal_new; /* lb-sec^2, BIN 0 */
static int dos_filtered_old;
static int dos_offset;
static uint sync_delta_timer;
static uchar dis_filter_latch;
static uint input_spd_first_point;
static int eng_decel_latest;
static int eng_decel_rate;
static int eng_decel_rate_wth_jake;
static uchar used_eng_brk;

/* shiftability calculations */

#define TORQ_ACC_LPF 230 /* 0.9000 BIN 8 */
#define K7_ABILITY 249 /* 0.9730 BIN 8 */
#define FIXED_LIMIT -30
#define FIXED_LIMIT_1 -36
#define SHF_BLY_HOLD_COUNT 3 /* 120 ms a 0.040 period */

static uchar shiftability_hold;
static uchar shiftability_hold_11;
static uchar shf_bly_hold_cntr;
static int dos_predicted_raw; /* BIN 0 */
static long dos_predicted_bin8; /* BIN 8 */
static long dos_predicted_partial_1; /* BIN 8 */
static long dos_predicted_partial_2; /* BIN 8 */
static int dos_prtd_lim_wth_jake; /* BIN 0 */
static int vehicle_accel_bs; /* BIN 0 */
static int vehicle_accel_as; /* BIN 0 */
static int engine_torque; /* BIN 0 */
static int torque_at_wheels; /* BIN 0 */
static int torq_to_accel_eng; /* BIN 0 */
static uint gcw_cal; /* lb-sec^2, BIN 0 */
static uint TRANS_STEP_SIZE_CAL; /* #, BIN 8 */
static uchar torque_accessories; /* lb-ft, BIN 0 */
#endif

#pragma EJECT

```

```

*****
* Function: mda_output_filter
*
* Description:
*   This is a one pole LPF with a variable coefficient. The magnitude
*   of the coefficient is directly related to the acceleration content
*   of the speed sample and the frequency.
*
*****
```

```

static void mda_output_filter(void)
{
    #define K1 8
    #define K2 24
    #define K3 48
    #define K4 160

    static register uint os_delta_speed;
    static register uchar weight;

    if (lpf_output_speed > output_speed)
        os_delta_speed = lpf_output_speed - output_speed;
    else
        os_delta_speed = output_speed - lpf_output_speed;

    if (os_delta_speed <= K1)          /* delta <= 200 rpm/s */
    {
        if (w1 > 1) --w1;
        if (w2 < 5) ++w2;
        if (w3 < 6) ++w3;
        if (w4 < 7) ++w4;
        weight = w1;
    }
    else if (os_delta_speed <= K2)    /* 200 rpm/s < delta <= 600 rpm/s */
    {
        if (w1 < 4) ++w1;
        if (w2 > 2) --w2;
        if (w3 < 6) ++w3;
        if (w4 < 7) ++w4;
        weight = w2;
    }
    else if (os_delta_speed <= K3)    /* 600 rpm/s < delta <= 1200 rpm/s */
    {
        if (w1 < 4) ++w1;
        if (w2 < 5) ++w2;
        if (w3 > 3) --w3;
        if (w4 < 7) ++w4;
        weight = w3;
    }
    else if (os_delta_speed <= K4)    /* 1200 rpm/s < delta <= 4000 rpm/s */
    {
        if (w1 < 4) ++w1;
        if (w2 < 5) ++w2;
        if (w3 < 6) ++w3;
        if (w4 > 3) --w4;
        weight = w4;
    }
    else                                /* 4000 rpm/s < delta */
    {
        if (w1 < 4) ++w1;
        if (w2 < 5) ++w2;
        if (w3 < 6) ++w3;
        if (w4 < 7) ++w4;
        weight = 7;
    }

    lpf_output_speed = lpf_output_speed +
        (output_speed >> weight) - (lpf_output_speed >> weight);
}

#pragma EJECT

```

```
*****
* Function: new_input_speed
*
* Description:
*   A utility function that returns the input shaft speed expected if
*   "gear" was engaged with the present output shaft speed.
*****
static uint new_input_speed(char gear)
{
    /* new_input_speed = lpf_output_speed * gear_ratio */
    _ax = trn_tbl.gear_ratio[gear + GR_OFS];           /* BIN 8 */
    asm mulu _axbx, lpf_output_speed;                  /* BIN 0+8=8 (_axbx) */
    asm shr1 _axbx, #8;                                /* BIN 8>>=0 (_ax) */
    return _ax;
}

#pragma EJECT
```

```
*****
* Function: auto_upshift
*
* Description:
*   This boolean function returns true when automatic upshift
*   conditions have been met.
*****
static int auto_upshift(void)
{
    if ((gos > upshift_point) && (output_speed_filtered > 120))
    {
        if (engine_communication_active)
        {
            /* if (!(shiftability_hold) && !(shiftability_hold_II)) */
            return 1;
        }
    }
    return 0;
}

#pragma EJECT
```

```
*****  
*  
* Function: auto_downshift  
*  
* Description:  
* This boolean function returns true when automatic downshift  
* conditions have been met.  
*  
*****/  
  
static int auto_downshift(void)  
{  
    if (gos < downshift_point)  
    {  
        return 1;  
    }  
    return 0;  
}  
#pragma EJECT
```

```
*****
* Function: determine_autosplit_type
*
* Description:
*   This function is used to determine if the impending shift type is
*   MANUAL or AUTO.
*
*****
```

```
static char determine_autosplit_type(char passed_new_gear, char passed_initial_gear)
{
    register char new_gr = passed_new_gear;
    register char init_gr = passed_initial_gear;

    if ((shift_in_process == FALSE) || (engine_status == ENGINE_RECOVERY_MODE))
    {
        if ((new_gr == 1 && init_gr == 2) || /* dn */
            (new_gr == 3 && init_gr == 4) || /* dn */
            (new_gr == 5 && init_gr == 6) || /* dn */
            (new_gr == 7 && init_gr == 8) || /* dn */
            (new_gr == 9 && init_gr == 10) || /* dn */
            (new_gr == 10 && init_gr == 9) || /* up */
            (new_gr == 8 && init_gr == 7) || /* up */
            (new_gr == 6 && init_gr == 5) || /* up */
            (new_gr == 4 && init_gr == 3) || /* up */
            (new_gr == 2 && init_gr == 1)) /* up */
            shift_init_type = AUTO;
        else
            shift_init_type = MANUAL;
    }
}
```

```
#pragma EJECT
```

```

/*
 * Function: getAutomaticGear
 *
 * Description:
 *   This function determines the appropriate gear for both automatic
 *   splitter shifts and manual lever shifts.
 */
static char getAutomaticGear(char initial_gear, char manual_request)
{
    register char new_gear = initial_gear;

    if (automatic_sip != -1)
    {
        /* initiate or continue an upshift: search up from lowest_forward
         (fastest input speed) for the first available gear that will provide input
         speed below a value (approx upshift rpm, minus an offset for gears that will
         result in a net downshift) */
        for (new_gear = lowest_forward;
            (new_input_speed(new_gear) > (upshift_point -
                (new_gear < initial_gear ?
                    (shf_tbl.up_offset_rpm + auto_dn_offset_rpm) : shf_tbl.best_gr_offset))) &&
            (new_gear < trn_tbl.highest_forward);
            ++new_gear);

        if ((initial_gear == 3) && ((new_gear == 2) || (new_gear == 1)))
            new_gear = initial_gear;

        desired_gear = new_gear;
        desired_gear_up = new_gear;
        determine_autosplit_type(new_gear, initial_gear);

        /* if lever shift and still in gear, keep initial_gear */
        if (((shift_init_type == MANUAL) && (transmission_position == IN_GEAR)) ||
            ((automatic_sip == 0) && (new_gear == initial_gear)) ||
            ((dgos < SHIFT_INHIBIT_DECEL_LIMIT) &&
            ((transmission_position == IN_GEAR) ||
            ((transmission_position == OUT_OF_GEAR) && (shift_init_type == AUTO))))))
            new_gear = initial_gear;
        else
        {
            /* indicate gear change and adjust downshift_point */
            automatic_sip = +1;
            auto_up_offset_rpm = 0;
            if (shift_init_type == AUTO)
                auto_dn_offset_rpm = shf_tbl.dwn_timer_offset_rpm;
            else
                auto_dn_offset_rpm = 0;
        }
    }

    if ((automatic_sip != 1) && (initial_gear > lowest_forward))
    {
        /* initiate or continue an downshift: search down from
         highest_forward (slowest input speed) for the first available gear that will
         provide input speed above a value (approx downshift rpm, plus an offset for
         gears that will result in a net upshift) */
        for (new_gear = trn_tbl.highest_forward;
            (new_input_speed(new_gear) < (downshift_point +
                (new_gear > initial_gear ? shf_tbl.dwn_offset_rpm : shf_tbl.best_gr_offset))) &&
            (new_gear > lowest_forward);
            --new_gear);

        if ((initial_gear == 3) && ((new_gear == 2) || (new_gear == 1)))
            new_gear = initial_gear;

        desired_gear_dn = new_gear;

        if (desired_gear_dn < initial_gear) /* Must be a down shift or else it may */
            desired_gear = new_gear; /* wrongly cancel the desired_up pick. */
    }
}

```

```
determine_automaplit_type(new_gear, initial_gear);

/* if lever shift and still in gear, keep initial_gear */
if (((shift_init_type == MANUAL) && (transmission_position == IN_GEAR)) ||
    ((automatic_sip == 0) && (new_gear >= initial_gear)) ||
    ((dgos < SHIFT_INHIBIT_DECEL_LIMIT) &&
     ((transmission_position == IN_GEAR) ||
      ((transmission_position == OUT_OF_GEAR) && (shift_init_type == AUTO)))))
    new_gear = initial_gear;
else
{
    /* indicate automatic gear change and adjust upshift_point */
    automatic_sip = -1;
    auto_dn_offset_rpm = 0;
    if (shift_init_type == AUTO)
        auto_up_offset_rpm = shf_tbl.up_timer_offset_rpm;
    else
        auto_up_offset_rpm = 0;
}
return new_gear;
}

#pragma EJECT
```

```
*****  
*  
* Function: determine_destination  
*  
* Description:  
*   This function uses "coasting_latch" to determine if a coasting or  
*   skip shift is being attempted. When sensed, the latch is used in  
*   the determine_base_pts function to affect the base shift points.  
*  
*****  
  
void determine_destination(void)  
{  
    /* if coasting in neutral - modify shift points */  
  
    if (coasting_latch == FALSE)  
    {  
        if ((last_known_gear - destination_gear_selected) > 1) /* multi downshift */  
        {  
            if ((destination_gear_selected == 7) ||  
                (destination_gear_selected == 5) ||  
                (destination_gear_selected == 3) ||  
                (destination_gear_selected == 1))  
            {  
                destination_gear_selected++;  
                coasting_latch = TRUE;  
            }  
        }  
        else  
        {  
            if ((destination_gear_selected - last_known_gear) > 1) /* multi upshift */  
            {  
                if ((destination_gear_selected == 10) ||  
                    (destination_gear_selected == 8) ||  
                    (destination_gear_selected == 6) ||  
                    (destination_gear_selected == 4))  
                {  
                    destination_gear_selected--;  
                    coasting_latch = TRUE;  
                }  
            }  
        }  
    }  
    else  
        if (shift_in_process == FALSE)  
            coasting_latch = FALSE;  
    }  
  
    #pragma EJECT
```

```
*****
* Function: determine_manual_shift_pts
*
* Description:
*
*****
```

```
static void determine_manual_shift_pts(void)
{
    if (coasting_latch == FALSE)
    {
        if (((last_known_gear == 1) ||
             (last_known_gear == 3) ||
             (last_known_gear == 5) ||
             (last_known_gear == 7) ||
             (last_known_gear == 9)))
            auto_dn_rpm = 1325; /* manual downshifts */

        else
            auto_up_rpm = 1425; /* manual upshifts */
    }
}
```

```
#pragma EJECT
```

```

*****
* Function: determine_base_auto_shift_pts
*
* Description:
*   This function determines the base up and down shift points based on
*   the position of the throttle. These base points will be used in the
*   calculation of the upshift_point and the downshift_point.
*
*   The anti-hunting calculations have been moved to this function since
*   these calculations are now throttle dependent.
*
*****
```

```

static void determine_base_auto_shift_pts(void)
{
    if (pct_demand_at_cur_sp > 0)
    {
        /* auto_up_rpm = shf_tbl.auto_up_lo_base +
           ((shf_tbl.aut_up_rpm - shf_tbl.auto_up_lo_base) * %throttle) */

        _cx = shf_tbl.aut_up_rpm - shf_tbl.auto_up_lo_base;
        _bx = pct_demand_at_cur_sp;
        asm mulu_cwdx, _bx;
        asm divu_cwdx, #100;
        auto_up_rpm = shf_tbl.auto_up_lo_base + _cx;

        /* check for RTD requirement */
        if (pct_demand_at_cur_sp > 90)
            auto_up_rpm += shf_tbl.auto_rtd_offset;

        /* auto_dn_rpm = shf_tbl.auto_dn_lo_base +
           ((shf_tbl.aut_dwn_rpm - shf_tbl.auto_dn_lo_base) * %throttle) */

        _cx = shf_tbl.aut_dwn_rpm - shf_tbl.auto_dn_lo_base;
        _bx = pct_demand_at_cur_sp;
        asm mulu_cwdx, _bx;
        asm divu_cwdx, #100;
        auto_dn_rpm = shf_tbl.auto_dn_lo_base + _cx;
    }
    else
    {
        auto_up_rpm = shf_tbl.auto_up_lo_base;
        auto_dn_rpm = shf_tbl.auto_dn_lo_base;
    }

    determine_manual_shift_pts();

    if (shift_in_process)
    {
        /* reset antihunt_counter */
        antihunt_counter = 0;

        /* allow the knob display to flashed any new desired gear */
        flash_desired_allowed = TRUE;
    }
    else
    {
        /* reset shift in process flags and update antihunt_counter */
        automatic_sip = 0;
        if (antihunt_counter < 255)
        {
            ++antihunt_counter;
        }

        /* look for upshift anti-hunt reset conditions */
        if ((antihunt_counter * (US_PER_LOOP/1000)) >= shf_tbl.offset_time)
        {
            /* check for last shift = upshift effects */
            if (auto_dn_offset_rpm == shf_tbl.dwn_timer_offset_rpm)
                auto_dn_offset_rpm = shf_tbl.dwn_offset_rpm;
            else if ((auto_dn_offset_rpm == shf_tbl.dwn_offset_rpm) &&
                     (input_speed_filtered > auto_dn_rpm + shf_tbl.dwn_reset_rpm))
                auto_dn_offset_rpm = 0;

            /* check for last shift = downshift effects */
            if (auto_up_offset_rpm == shf_tbl.up_timer_offset_rpm)
                auto_up_offset_rpm = shf_tbl.up_offset_rpm;
        }
    }
}
```

```
else if ((auto_up_offset_rpm == shf_tbl.up_offset_rpm) &&
        (input_speed_filtered > auto_up_rpm - shf_tbl.up_reset_rpm))
    auto_up_offset_rpm = 0;

/* allow the knob display to flashed the desired gear */
if (((desired_gear > destination_gear_selected) && (gos < (upshift_point + 25))) ||
    ((desired_gear < destination_gear_selected) && (gos > (downshift_point - 25))))
    flash_desired_allowed = FALSE;
else
    flash_desired_allowed = TRUE;
}

/* set the shift points based on throttle and determined offsets */
upshift_point = auto_up_rpm + auto_up_offset_rpm;
downshift_point = auto_dn_rpm - auto_dn_offset_rpm;

/* check that the calculated shift point is reasonable */
if (upshift_point > shf_tbl.man_dwn_sync_rpm)
    upshift_point = shf_tbl.man_dwn_sync_rpm;

if (downshift_point < shf_tbl.man_up_sync_rpm)
    downshift_point = shf_tbl.man_up_sync_rpm;

}

#pragma EJECT
```

```

*****+
* Function: select_gear
*
* Description:
*   This is the root function for the periodic task SELECT_GEAR. Each
*   loop begins by checking the manual up/down buttons. Then, based on
*   selected_mode and output shaft speed, a 'get_..._gear' function is
*   called to update destination_gear_selected.
*
*****+
void select_gear(void)
{
    char manual_request;          /* current manual request (+/- 1) */
    static uchar enable_gcw_calc; /* diagnostic - delete later !!*/
    enable_gcw_calc = FALSE;      /* diagnostic - delete later !!*/
    shf_tbl = ini_shf_tbl;        /* initialize the shift table */

    destination_gear_selected = 1;
    desired_gear = 1;

    /* initialize file scope variables */

    w1 = 3;
    w2 = 4;
    w3 = 5;
    w4 = 6;
    lpf_output_speed = output_speed;

    upshift_point = shf_tbl.aut_up_rpm;
    downshift_point = shf_tbl.aut_dwn_rpm;
    auto_up_offset_rpm = shf_tbl.up_timer_offset_rpm;
    auto_dn_offset_rpm = shf_tbl.dwn_timer_offset_rpm;
    lowest_forward = INITIAL_START_GEAR;
    automatic_sip = 0;
    antihunt_counter = 255U;
    coasting_latch = FALSE;
    flash_desired_allowed = TRUE;

#if (0)      /* shiftability variables */
    zero_speed_counter = ZERO_SPEED_TIME_LIMIT + 5; /* force init of gcw variables */
    eng_decel_rate = -400; /* RPM/SEC */
#endif

    TRANS_STEP_SIZE_CAL = 346; /* 1.352 BIN 8 (was constant) */
    torque_accessories = 100; /* accessory load with air condition off in coach */
    gcw_cal = 2400;           /* (GCW x 1.68)/32.17 BIN 0 (was constant) */
    gcw_cal_new = 2400;        /* (GCW x 1.68)/32.17 BIN 0 (was constant) */
    gcw_local = 45000;
    gcw_local_counter = 50;
    gcw_local_total = 2250000;
    gross_combined_weight = 45000;

#endif      /* shiftability variables */

    x_start_periodic();
    while (1)
    {
        mda_output_filter(); /* update our filtered output speed */

#if (0)      /* shiftability */
        if (enable_gcw_calc) /* diagnostic aid !!*/
        {
            determine_gross_combined_weight();
            determine_shiftability();
            determine_shiftability_ll();
        }
        /* shiftability */
#endif
        manual_request = 0;
        determine_base_auto_shift_pts();

        /* set destination_gear_selected from function(s) appropriate for selected_mode */
        switch(selected_mode)
        {
            case REVERSE_MODE:

```

```
if ((forward_left == TRUE) &&
    (downshift_delayed == FALSE) &&
    (low_speed_latch == FALSE))
{
    destination_gear_selected = get_automatic_gear(destination_gear_selected, manual_request);
    determine_destination();
}
break;

case LOW_MODE:
case HOLD_MODE:
case NEUTRAL_MODE:
case PARK_MODE:
case POWER_UP_MODE:
case POWER_DOWN_MODE:
case DIAGNOSTIC_TEST_MODE:
    /* prevent transient selection upon mode change (these modes ignore it) */
    destination_gear_selected = 0;
    break;

default:
    /* invalid mode: do nothing */
    break;
}
x_sync_periodic(US_PER_LOOP);
}
x_end_periodic();
```

```
*****
* * Unpublished and confidential. Not to be reproduced,
* disseminated, transferred or used without the prior
* written consent of Eaton Corporation.
* *
* Copyright Eaton Corporation, 1994.
* All rights reserved.
*****
* Filename: trns_act.c96 (R-747) (AutoSplit)
*
* Description:
* This module monitors and controls the transmission's actions.
*
* Part Number: <none>
*
* Rev 1.5 12 Dec 1994 09:19 markyvech
* In "determine_gear" function; added condition for "intent_hold" and
* "in_gear_switch".
*
* Rev 1.4 9 Dec 1994 14:25 markyvech
* In "determine_gear" function; added conditions that the "gear_in_timer"
* will be held high while in the engine_sync_mode and the intent_to_shift
* switch is depressed. (A smaller sync error is maintained for easier
* lever engagement when the switch is depressed.) Also changed the offset
* for clearing the low_speed_latch from downshift_point + 100 to
* downshift_point + 275.
*
* Rev 1.3 9 Dec 1994 07:59 markyvech
* In "determine_splitter_state_autosplit" function; added the conditions
* to preselect the splitter if the "intent_to_shift_switch" is TRUE.
*
* Rev 1.2 7 Dec 1994 15:53 markyvech
* Added the "determine_range_state" function. Set the appropriate range
* solenoid select flag based on the electric range select switch.
*
* Rev 1.1 5 Dec 1994 14:54 markyvech
* Broke the "determine_splitter_state" function into multiple functions
* based on the type of base transmission system we are working with.
*
* Rev 1.0 3 May 1994 13:35:04 markyvech
* Initial revision.
*****
/* Header files included.
*/
*****
```



```
#include <exec.h> /* executive information */
#include <c_regs.h> /* KR internal register definitions */
#include <kr_sfr.h> /* defines the kr special function registers */
#include <kr_def.h> /* 80c196kr bits, constants, and structures */
#include <kwslib.h>
#include "drl_cmds.h" /* engine control interface */
#include "trns_act.h" /* interface to this module */
#include "trn_tbl.h" /* transmission table */
#include "calc_spd.h"
#include "cont_sys.h"
#include "sel_gear.h"
#include "conj1939.h"
#include "pr_s_i_s.h"

*****
* Variables declared by this file.
*****
register unsigned char transmission_position;

unsigned char R747_type;
unsigned char jammed_in_wrong_gear_timer;
unsigned char manual_sync_delayed_timer;
unsigned char low_speed_latch;
unsigned char forward_last;
```

```
unsigned char splitter_hi;
unsigned char splitter_lo;
unsigned char range_hi;
unsigned char range_lo;
unsigned char splitter_timer;
unsigned char splitter_within_sync;
unsigned char aux_box;
unsigned char downshift_delayed;
    signed char g_ptr_old;
    signed char current_gear;
    signed char last_known_gear;
unsigned int gear_in_timer;
unsigned int gear_out_timer;
unsigned int normal_auto_neutral_timer;
unsigned int abs_trans_sync_error;
unsigned int trans_window_calc;
    signed int input_speed_modified;
    signed int trans_sync_error;
    signed int range_error;
    signed int range_cal;
    signed int splitter_tc;
    signed long isdgc;
    signed long gros;
    signed char g_ptr;
```

```
#pragma EJECT
```

```

*****
* #defines and constants local to this file.
*
*****
```

```

#define US_PER_LOOP 10000U
#define JAMMED_TIME 200      /* 2.0 SECONDS */

static const uchar SPLITTER_LO_TABLE[23] =
{
    0,    /* -4 */
    0,    /* -3 */
    ON,   /* -2 */ /* split_lo = OFF, split_hi = ON; overdrive */
    ON,   /* -1 */ /* split_lo = ON, split_hi = OFF; direct */
    ON,   /* 0 */  /* split_lo = ON, split_hi = OFF; direct */
    ON,   /* 1 */  /* split_lo = ON, split_hi = OFF; direct */
    ON,   /* 2 */  /* split_lo = OFF, split_hi = ON; overdrive */
    ON,   /* 3 */  /* split_lo = ON, split_hi = OFF; direct */
    ON,   /* 4 */  /* split_lo = OFF, split_hi = ON; overdrive */
    ON,   /* 5 */  /* split_lo = ON, split_hi = OFF; direct */
    ON,   /* 6 */  /* split_lo = OFF, split_hi = ON; overdrive */
    ON,   /* 7 */  /* split_lo = ON, split_hi = OFF; direct */
    ON,   /* 8 */  /* split_lo = OFF, split_hi = ON; overdrive */
    ON,   /* 9 */  /* split_lo = ON, split_hi = OFF; direct */
    ON,   /* 10 */ /* split_lo = OFF, split_hi = ON; overdrive */
    0,    /* 11 */
    0,    /* 12 */
    0,    /* 13 */
    0,    /* 14 */
    0,    /* 15 */
    0,    /* 16 */
    0,    /* 17 */
    0     /* 18 */
};

static const uchar SPLITTER_HI_TABLE[23] =
{
    0,    /* -4 */
    0,    /* -3 */
    ON,   /* -2 */ /* split_lo = OFF, split_hi = ON; overdrive */
    OFF,  /* -1 */ /* split_lo = ON, split_hi = OFF; direct */
    OFF,  /* 0 */  /* split_lo = ON, split_hi = OFF; direct */
    OFF,  /* 1 */  /* split_lo = ON, split_hi = OFF; direct */
    ON,   /* 2 */  /* split_lo = OFF, split_hi = ON; overdrive */
    OFF,  /* 3 */  /* split_lo = ON, split_hi = OFF; direct */
    ON,   /* 4 */  /* split_lo = OFF, split_hi = ON; overdrive */
    OFF,  /* 5 */  /* split_lo = ON, split_hi = OFF; direct */
    ON,   /* 6 */  /* split_lo = OFF, split_hi = ON; overdrive */
    OFF,  /* 7 */  /* split_lo = ON, split_hi = OFF; direct */
    ON,   /* 8 */  /* split_lo = OFF, split_hi = ON; overdrive */
    OFF,  /* 9 */  /* split_lo = ON, split_hi = OFF; direct */
    ON,   /* 10 */ /* split_lo = OFF, split_hi = ON; overdrive */
    0,    /* 11 */
    0,    /* 12 */
    0,    /* 13 */
    0,    /* 14 */
    0,    /* 15 */
    0,    /* 16 */
    0,    /* 17 */
    0     /* 18 */
};

#endif

```

```
#pragma EJECT
```

```
static const uchar SPLITTER_TC_TABLE[23] =
{
    /* Splitter movement time constant in milliseconds */
    0,      /* -4 */
    0,      /* -3 */
    100,   /* -2 */ /* splitter = overdrive */
    100,   /* -1 */ /* splitter = direct */
    100,   /*  0 */ /* splitter = direct */
    100,   /*  1 */ /* splitter = direct */
    100,   /*  2 */ /* splitter = overdrive */
    100,   /*  3 */ /* splitter = direct */
    100,   /*  4 */ /* splitter = overdrive */
    100,   /*  5 */ /* splitter = direct */
    100,   /*  6 */ /* splitter = overdrive */
    100,   /*  7 */ /* splitter = direct */
    100,   /*  8 */ /* splitter = overdrive */
    100,   /*  9 */ /* splitter = direct */
    100,   /* 10 */ /* splitter = overdrive */
    0,      /* 11 */
    0,      /* 12 */
    0,      /* 13 */
    0,      /* 14 */
    0,      /* 15 */
    0,      /* 16 */
    0,      /* 17 */
    0       /* 18 */
};

#endif
```

```
#pragma EJECT
```

```
*****
* Function: Initialize_Trans_Action
*
* Description:
*   This function initializes those module variables that must be set to a
*   known state on power up or reset.
*
*****
```

```
void initialize_trans_action(void)
{
    gear_in_timer = 500;
    gear_out_timer = 500;
    g_ptr_old = 0;
    current_gear = 0;
    last_known_gear = 0;
    destination_gear = 0;
    transmission_position = OUT_OF_GEAR;
    low_speed_latch = TRUE;
    splitter_lo = 0;
    splitter_hi = 0;
    range_lo = 0;
    range_hi = 0;
    normal_auto_neutral_timer = 0;
    downshift_delayed = FALSE;
    manual_sync_delayed_timer = 0;
    jammed_in_wrong_gear_timer = JAMMED_TIME;
    R747_type = INTENT;
}
#pragma EJECT
```

```
*****
* Function: Check_For_Jammed_Gear
*
* Description:
*   This function checks for a indicated engaged gear, (g_ptr), that is different
*   than "destination_gear_selected" and will force the needed conditions to
*   recover. This condition can occur when the transmission is "jammed" into
*   the wrong lever position, (usually with the clutch disengaged), during a
*   shift.
*****
void check_for_jammed_gear(void)
{
    if ((shift_in_process == TRUE) &&
        (automatic_sip != 0) &&
        (destination_gear_selected != g_ptr) &&
        (g_ptr != 0) &&
        (jammed_in_wrong_gear_timer > 0))
        jammed_in_wrong_gear_timer--;

    if (jammed_in_wrong_gear_timer == 0)
    {
        shift_in_process = FALSE;
        automatic_sip = 0;
        desired_gear = current_gear;
        destination_gear = current_gear;
        destination_gear_selected = current_gear;
        jammed_in_wrong_gear_timer = JAMMED_TIME;
        engine_commands = ENGINE_RECOVERY;
    }
}

#endif

#pragma EJECT
```

```

*****
* Function: determine_gear
*
* Description:
* This function determines the current gear that the transmission
* is in. When conditions are such that the current gear can not be
* determined it will be set to a default, (0).
*
* Note: When the error across the transmission is near zero for some
* time for a given test gear then it will be deemed in that gear.
*
*     trans_sync_error = input_spd/gf[gear] - gr[gear] * os
*
*****
```

void determine_gear(void)

```

#define BIN_8          256 /* 2 BIN 8 */
#define MAX_ERR        4000 /* RPM */
#define WINDOW         30 /* 30 RPM */
#define GEAR_IN_TIME_LEVER 125 /* 250 MSEC a 5 cnts per loop */
#define GEAR_IN_TIME_AUTO   50 /* 100 MSEC a 5 cnts per loop */
#define GEAR_OUT_TIME      8 /* 80 MSEC */
#define MANUAL_SYNC_DELAYED_TIME 45 /* 450 MSEC */
#define ERROR_FUDGE_FACTOR    3 /* RPM */
#define NORMAL_AUTO_TIME     250 /* 2.5 SEC */

#if (0)
    g_ptr = -1; /* lowest reverse ratio */

    /** isdgf = input_speed_filtered / front box gear ratio */
    _bx = (signed int)(input_speed_filtered);
    _cx = BIN_8;
    _ax = trn_tbl.GF[g_ptr + GR_OFS];
    asm mul _cxdx, _bx;
    asm div _cxdx, _ax;
    isdgf = _cx;

    /** gros = output_speed_filtered * rear box ratio */
    _bx = (signed int)(output_speed_filtered + ERROR_FUDGE_FACTOR);
    _cx = trn_tbl.GR[g_ptr + GR_OFS];
    _ax = BIN_8;
    asm mul _cxdx, _bx;
    asm div _cxdx, _ax;
    gros = _cx;

    trans_sync_error = (isdgf - gros);
    if (isdgf > gros)
        abs_trans_sync_error = (unsigned int)(isdgf - gros);
    else
        abs_trans_sync_error = (unsigned int)(gros - isdgf);
#endif

    abs_trans_sync_error = MAX_ERR;
    trans_window_calc = 0;

    if (abs_trans_sync_error > trans_window_calc) /* if not in reverse, check for forward */
    {
        g_ptr = 1 + trn_tbl.highest_forward;
        abs_trans_sync_error = MAX_ERR;
        while ((abs_trans_sync_error > trans_window_calc) && (g_ptr != 0))
        {
            g_ptr--;
            /** isdgf = input_speed_filtered / front box gear ratio */
            _bx = (signed int)(input_speed_filtered);
            _cx = BIN_8;
            _ax = trn_tbl.GF[g_ptr + GR_OFS];
            asm mul _cxdx, _bx;
            asm div _cxdx, _ax;
            isdgf = _cx;

            /** gros = output_speed_filtered * rear box ratio */
            _bx = (signed int)(output_speed_filtered + ERROR_FUDGE_FACTOR);
            _cx = trn_tbl.GR[g_ptr + GR_OFS];
            _ax = BIN_8;
            asm mul _cxdx, _bx;
            asm div _cxdx, _ax;
            gros = _cx;
        }
    }

```

```

trans_sync_error = isdgf - gros;
if (isdgf > gros)
    abs_trans_sync_error = (int)(isdgf - gros);
else
    abs_trans_sync_error = (int)(gros - isdgf);

/* calculate trans sync error window based on gear pointer */
_bx = WINDOW;                                /* BIN 0 */
_cx = BIN_8;                                  /* BIN 8 */
_ax = trn_tbl.GF[g_ptr + GR_OFS];           /* BIN 8 */
asm mulu _cxidx, _bx;                         /* make WINDOW BIN 8 */
asm divu _cxidx, _ax;                          /* divide by front ration BIN 8 */
trans_window_calc = _cx;                      /* BIN 0 */

}

if (g_ptr == 0)                                /* If in neutral, force values */
{
    abs_trans_sync_error = MAX_ERR;
    trans_sync_error = MAX_ERR;
    trans_window_calc = 0;
    isdgf = 0;
    gros = 0;
    downshift_delayed = FALSE;
    jammed_in_wrong_gear_timer = JAMMED_TIME; /* Initialize for next occurrence */
}

if ((abs_trans_sync_error > trans_window_calc) || /* Must have error for some */
    ((g_ptr != current_gear) && (current_gear != 0))) /* before neutral state is */
{                                                 /* recognized. */
    if (gear_out_timer == 0)
    {
        transmission_position = OUT_OF_GEAR;
        current_gear = 0;
    }

    else
        gear_out_timer--;
}
else
    gear_out_timer = GEAR_OUT_TIME;

if ((g_ptr != g_ptr_old) || (g_ptr == 0) ||
    ((intent_to_shift_switch == TRUE) && /* intent to shift stuff */
     (in_gear_switch == FALSE) &&
     (engine_commands == ENGINE_SYNC) &&
     (shift_init_type == MANUAL)) ||
    ((accelerator_pedal_position < 5) &&
     (input_speed < 700) &&
     (low_speed_latch == FALSE)))
{
    if ((engine_commands == ENGINE_SYNC) ||
        (engine_commands == ENGINE_PREDIP))
    {
        if (engine_commands == ENGINE_PREDIP)          /* Use a short in gear time */
        {
            normal_auto_neutral_timer = 0;
            manual_sync_delayed_timer = 0;
        }

        else
            if (normal_auto_neutral_timer <= NORMAL_AUTO_TIME)
                normal_auto_neutral_timer++;
    }

    if ((shift_init_type == AUTO) &&
        (normal_auto_neutral_timer < NORMAL_AUTO_TIME))
        gear_in_timer = GEAR_IN_TIME_AUTO;
    else
        gear_in_timer = GEAR_IN_TIME_LEVER;
}

else
    if (gear_in_timer == 0)
    {
        current_gear = g_ptr;
    }
}

```

```

last_known_gear = g_ptr;
transmission_position = IN_GEAR;
normal_auto_neutral_timer = 0; /* get ready for next time */
downshift_delayed = FALSE;

if ((intent_to_shift_switch == TRUE) && (engine_commands == ENGINE_SYNC))
    intent_hold = TRUE;

if ((gos_current_gear > (downshift_point + 275)) &&
    (low_speed_latch == TRUE))
{
    low_speed_latch = FALSE;
    destination_gear = current_gear;
    destination_gear_selected = current_gear;
    desired_gear = current_gear;
    lowest_forward = current_gear;
}
else
{
    if (low_speed_latch == TRUE)
    {
        destination_gear = lowest_forward; /* was set to 1 */
        destination_gear_selected = lowest_forward; /* was set to 1 */
        desired_gear = lowest_forward; /* was set to 1 */
        shift_in_process = FALSE;
    }
}

if (last_known_gear > 0) /* Record REV/FOR data for */
    forward_last = TRUE; /* use in the select_gear */
else
    forward_last = FALSE;
}

else
{
    if (((shift_init_type == AUTO) || (low_speed_latch == TRUE) ||
        (manual_sync_delayed_timer >= MANUAL_SYNC_DELAYED_TIME))) &&
        (gear_in_timer >= 4))
        gear_in_timer -= 4;

    if (gear_in_timer > 0)
        gear_in_timer--;

    if (destination_gear_selected == g_ptr) /* Prevent splitter shift while */
        downshift_delayed = TRUE; /* still confirming a lever shift. */
}

check_for_jammed_gear();

g_ptr_old = g_ptr;

if (((engine_commands == ENGINE_SYNC) || /* manual_sync_delayed_timer is used to allow the sync */
    (engine_commands == ENGINE_FOLLOWER)) && /* mode's first pass a chance to pass the engine speed */
    (manual_sync_delayed_timer < MANUAL_SYNC_DELAYED_TIME)) /* thru sync and give the mechanicals a chance to */
    manual_sync_delayed_timer++; /* transition. At this time a false in_gear signal */
/* should be avoided. */

if (output_speed_filtered < 125) /* If stopped; set low speed latch */
{
    current_gear = 0;
    transmission_position = OUT_OF_GEAR;
    low_speed_latch = TRUE;

    if (splitter_launch_state == LO_SPLITTER)
    {
        if ((lowest_forward == 2) ||
            (lowest_forward == 4) ||
            (lowest_forward == 6))
            lowest_forward--;
    }
    else
    {
        if ((lowest_forward == 1) ||
            (lowest_forward == 3) ||
            (lowest_forward == 5))
            lowest_forward++;
    }
}
}

```

~~Sprague EJECT~~

```
*****
* Function: determine_range_status
*
* Description:
* This function determines the status the of range.
*
* rng_err = rear_counter_spd - (range_ratio * output_spd)
*
* rcs = 54/21 * 44 * os (for low range)
*
* rcs = 42/51 * 44 * os (for high range)
*
*****/
```

```
void determine_range_status(void)
{
#define BIN_12          4096 /* 2 bin 12 */
#define HI_RANGE_GEAR    7
#define LO_RANGE_CAL    10532 /* 54/21 BIN 12 */
#define HI_RANGE_CAL   3373 /* 42/51 BIN 12 */
#define RANGE_WINDOW_POS 30 /* 30 RPM */
#define RANGE_WINDOW_NEG -30 /* -30 RPM */

/** This code was never tested or used during the concept development ***
/** phase. However, it is likely to be needed for range protection ***
/** in the product. ***

if (destination_gear >= HI_RANGE_GEAR)
    range_cal = HI_RANGE_CAL;
else
    range_cal = LO_RANGE_CAL;

range_error =(((aux_speed * BIN_12)
              - (range_cal * output_speed_filtered))/BIN_12);

if ((range_error > RANGE_WINDOW_POS) || (range_error < RANGE_WINDOW_NEG))
    aux_box = OUT_OF_GEAR;
else
    aux_box = IN_GEAR;

}

#endif
```

```
*****
* Function: determine_range_state
*
* Description:
* This function determines the correct state for the range.
*
*****
```

```
void determine_range_state(void)
{
    if (range_select_switch == HIGH)
    {
        range_lo = OFF;      /* Turn off the low range solenoid */
        range_hi = ON;       /* Turn on the high range solenoid */
    }
    else
    {
        range_lo = ON;       /* Turn on the low range solenoid */
        range_hi = OFF;      /* Turn off the high range solenoid */
    }
}
```

```
#pragma EJECT
```



```
*****
* Function: determine_splitter_state_dual_force
*
* Description:
* This function determines the correct state for the splitter.
*
*****
```

```
void determine_splitter_state_dual_force(void)
{
    if ((clutch_state == DISENGAGED) ||          /* use normal forces */
        (desired_gear > 6))
    {
        splitter_lo = ON;
    }
    else
    {
        if (splitter_select_switch == HIGH) /* use high force into overdrive */
            splitter_lo = OFF;
        else
            splitter_lo = ON;
    }

    if (splitter_select_switch == HIGH)
        splitter_hi = ON;
    else
        splitter_hi = OFF;
}
```

```
#pragma EJECT
```

```
*****
* Function: determine_splitter_state_base
* Description:
*   This function determines the correct state for the splitter.
*****
void determine_splitter_state_base(void)
{
    splitter_lo = ON;
    if (splitter_select_switch == HIGH)
        splitter_hi = ON;
    else
        splitter_hi = OFF;
}
#pragma EJECT
```

```
*****
* Function: determine_splitter_state
*
* Description:
* This function determines the correct state for the splitter.
*
*****
```

```
void determine_splitter_state(void)
{
    if (R747_type == BASE)
        determine_splitter_state_base();

    else
        if (R747_type == DUAL_FORCE)
            determine_splitter_state_dual_force();

    else
        determine_splitter_state_autosplit();
}

#pragma EJECT
```

```
*****
* Function: Transmission_Action
*
* Description:
*   This function controls the states of the system output devices.
*****
void transmission_action(void)
{
    initialize_trans_action();      /* initialize variables */

    x_start_periodic();
    while (1)
    {
        determine_gear();          /* calculate the current gear */
        determine_range_status();  /* determine range state */
        determine_range_state();   /* determine correct state for range */
        determine_splitter_state(); /* determine correct state for splitter */

        x_sync_periodic(US_PER_LOOP);
    }
    x_end_periodic();
}
```

1-11-95
Demos to
CCT
DOD

Files from C:\RON\R747\INTENT\INTENT.ZIP

DRL_CMDS.C96	12/09/94
PR_S_I_S.C96	12/12/94
SEL_GEAR.C96	12/12/94
SEQ_SHFT.C96	12/12/94
TRNS_ACT.C96	12/12/94

```

*****
*      Unpublished and confidential. Not to be reproduced,
*      disseminated, transferred or used without the prior
*      written consent of Eaton Corporation.
*
*      Copyright Eaton Corporation, 1993-94.
*      All rights reserved.
*****
*      Filename: drl_cmds.c96      (R-747) (AutoSplit)
*
*      Description:
*          The functions in this file will perform the required operations
*          for controlling driveline components on the J1939 communication link.
*
*      Part Number: <none>
*
*      Rev 1.4  09 Dec 1994 14:29      markyvech
*      In function "control_engine_sync"; added code to allow smaller sync
*      offsets if the intent to shift switch is depressed. This allows easier
*      shift lever insertion. (Note: the gear cannot be confirmed until the
*      switch is released.)
*
*      Rev 1.3  09 Dec 1994 11:18      markyvech
*      In function "control_engine_predit"; added code to allow recovery mode if
*      the intent switch is released before neutral is achieved. Also added the
*      pr_s_i_s.h to get the intent_to_shift_switch variable. Also added a faster
*      torque ramp off rate for manual intent to shifts
*
*      Rev 1.2  08 Dec 1994 14:44      markyvech
*      Changed the offsets in "control_engine_sync" from +-65 to +-45 RPM. Also
*      changed the input speed filter constant, (IS_FK1), from 236 to 235.
*
*      Rev 1.1  07 Dec 1994 15:33      markyvech
*      Took out the rear counter shaft speed substitution for output shaft speed
*      because there is no rear counter shaft speed sensor.
*
*      Rev 1.0  14 Sep 1994 10:48:40      markyvech
*      Initial revision.
*****

```

```

*****
*      Header files included.
*****

```

```

#include <exec.h>           /* executive information */
#include <c_regs.h>          /* KR internal register definitions */
#include <wwslib.h>           /* contains common global defines */
#include "cont_sys.h"         /* control system information */
#include "conj1939.h"          /* defines interface to j1939 control module */
#include "drl_cmds.h"          /* driveline commands information */
#include "trn_tbl.h"           /* transmission table data structures */
#include "sel_gear.h"          /* access to speed filter values */
#include "calc_spd.h"
#include "trns_act.h"
#include "pr_s_i_s.h"

```

```
#pragma noreentrant
```

```

*****
*      #defines local to this file.
*****

```

```
#define US_PER_LOOP 10000U
```

```
#define ACTIVE_RECOVERY_GEAR 10 /* rule out boosting downs for now */
```

```

*****
*      Constants and variables declared by this file.
*
```

```

*****
/* public */

register unsigned char engine_commands;
register unsigned char engine_status;

unsigned char desired_sync_test_mode;
unsigned char forced_predip;
unsigned int desired_engine_speed_test;
unsigned int desired_engine_speed_ramp;
unsigned char desired_engine_speed_timer;
unsigned char desired_engine_speed_time;
unsigned char eng_brake_command;
unsigned char eng_brake_assist;
unsigned char positive_pedal_trans;
unsigned char sync_first_pass_timer;
/* unsigned char clutch_state; */
unsigned int clutch_slip_speed;
signed int dos_filtered;
signed int overall_error;
unsigned int os_based_on_rcs;
unsigned int input_speed_filtered;
signed int dgos;
signed int input_speed_accel_filtered;
unsigned int output_speed_filtered;
unsigned long is_filtered_bin8;
unsigned long os_filtered_bin8;
signed long dis_filtered_bin8;

signed char eng_percent_torque_filtered;
signed char percent_torque_accessories;
signed char needed_percent_for_zero_flywheel_trq;
unsigned char zero_flywheel_trq_timer;
unsigned char zero_flywheel_trq_time;
unsigned char accelerator_pedal_position_old;
unsigned int gos; /* overall destination gear ratio * output speed BIN 0 */
signed int gos_signed; /* overall destination gear ratio * output speed BIN 0 */
signed int input_shaft_accel_calculated;

unsigned int gos_current_gear; /* overall current gear ratio * output speed BIN 0 */

unsigned char sync_first_pass;
unsigned int sync_maintain_timer;
signed int sync_offset;
signed int sync_offset_pos;
signed int sync_offset_neg;
signed int sync_dos_offset;
signed int sync_dos_offset_X1;
signed int sync_speed_modified;

/* local */

static unsigned int predip_timer_1;
static unsigned char predip_timer_2;
static unsigned char predip_timer_3;
static signed char predip_torque_bump_value;
static unsigned char predip_torque_bump_time;

static unsigned int sync_on_timer;
static unsigned int sync_off_timer;
static unsigned char sync_dither_timer;

static unsigned int torque_limit;
static unsigned char recovery_cancel_timer;
static unsigned int recov_coast_down_tmp1;
static unsigned int recov_coast_down_tmp2;

static signed int lpf_output_accel;

#endif

```

#pragma EJECT

```

*****
*          PREDIP MODE CONSTANTS
*
*****
```

#define PREDIP_ZERO_FDBK_TIME	40	/* 0.40s @10ms period */
#define PREDIP_TORQUE_ZERO_TIME	60	/* 0.60s @10ms period */
#define PREDIP_NORMAL_TIME	200	/* 2.00s @10ms period */
#define TORQUE_RAMP_OFF_RATE	1	/* 1% (per loop) */
#define PREDIP_TORQ_BUMP_VALUE_LO	0	/* 0% */
#define PREDIP_TORQ_BUMP_TIME_LO	15	/* 0.15s @10ms period */
#define PREDIP_TORQ_BUMP_VALUE_MED	10	/* 10% */
#define PREDIP_TORQ_BUMP_TIME_MED	25	/* 0.25s @10ms period */
#define PREDIP_TORQ_BUMP_VALUE_HI	25	/* 25% */
#define PREDIP_TORQ_BUMP_TIME_HI	30	/* 0.30s @10ms period */

```

*****
*          SYNC MODE CONSTANTS
*
*****
```

#define SYNC_DITHER_TIME_ABOVE	20	/* 0.20s @10ms period */
#define SYNC_DITHER_TIME_BELOW	30	/* 0.30s @10ms period */
#define SYNC_DITHER_RPM	35	/* 35 rpm */
#define SYNC_DITHER_FIRST_TIME	255	/* DUMMY VALUE */
#define MAINTAIN_SYNC_TIME	500	/* 5.00 Sec */
#define SYNC_FIRST_PASS_TIME	250	/* 2.50 Sec */
#define THREE_PERCENT	3	
#define ENG_RESPONSE_UPSHF_TIME	10	/* 10 msec */
#define ENG_RESPONSE_DNSHF_TIME	10	/* 10 msec */
#define SYNC_DOS_OFFSET_CONSTANT	2816	/* 11 BIN 8 */

```

*****
*          RECOVERY MODE CONSTANTS
*
*****
```

#define RECOVERY_CANCEL_TIME	10	/* 0.10s @10ms period */
#define RECOVERY_CANCEL_OFFSET	20	/* 20% BIN 0 */
#define RECOVERY_TORQUE_STEP	1280	/* 5% BIN 8 */
#define THLO_DS_ENG_DECAY_K1	450	
#define THLO_DS_ENG_DECAY_RAMP	1	/* 1 rpm BIN 0 */
#define THLO_DS_FINISHED_DELTA	200	/* 200 rpm BIN 0 */

```

static const uint RECOVERY_RATE_TABLE[23] =
{
    0,      /* -4 */
    0,      /* -3 */
    128,   /* -2 : 0.50% per loop BIN 8 */
    128,   /* -1 : 0.50% per loop BIN 8 */
    128,   /* 0 : 0.50% per loop BIN 8 */
    128,   /* 1 : 0.50% per loop BIN 8 */
    128,   /* 2 : 0.50% per loop BIN 8 */
    128,   /* 3 : 0.50% per loop BIN 8 */
    128,   /* 4 : 0.50% per loop BIN 8 */
    192,   /* 5 : 0.75% per loop BIN 8 */
    192,   /* 6 : 0.75% per loop BIN 8 */
    192,   /* 7 : 0.75% per loop BIN 8 */
    281,   /* 8 : 1.10% per loop BIN 8 */
    281,   /* 9 : 1.10% per loop BIN 8 */
    281,   /* 10 : 1.10% per loop BIN 8 */
    0,     /* 11 */
    0,     /* 12 */
    0,     /* 13 */
    0,     /* 14 */
    0,     /* 15 */
    0,     /* 16 */
    0,     /* 17 */
    0,     /* 18 */
};
```

```
*****
* Functions: initialize_driveline_data;
*
* Description:
*   This function, called after all resets, will initialize the system
*   copy of driveline related data received from the communications link.
*****
static void initialize_driveline_data(void)
{
    accelerator_pedal_position = 0;
    engine_communication_active = FALSE;
    engine_brake_available = FALSE;
    eng_brake_command = ENG_BRAKE_IDLE; /* should init with engine_commands */
/*    clutch_state = ENGAGED; */
    positive_pedal_trans = FALSE;
    zero_flywheel_trq_timer = 0;
    zero_flywheel_trq_time = 0;
    percent_torque_accessories = 3;

    desired_sync_test_mode = FALSE; /* debug use only - delete later */
    desired_engine_speed_test = 0; /* debug use only - delete later */
    desired_engine_speed_ramp = 0; /* debug use only - delete later */
    desired_engine_speed_timer = 0; /* debug use only - delete later */
    desired_engine_speed = 0;
    sync_dos_offset_K1 = SYNC_DOS_OFFSET_CONSTANT;
    desired_engine_speed_time = 0;
    forced_predip = FALSE;
}


```

```
#pragma EJECT
```

```

*****
* Function: control_engine_predip
*
* Description:
*   Determines throttle command for predip mode.
*
*   After a reasonable delay for the transmission to pull to neutral the
*   torque will be cycled from zero to a determined value to help the
*   transmission achieve neutral.
*
*****
```

```

static void control_engine_predip(void)
{
    if (engine_status != ENGINE_PREDIP_MODE)
    {
        engine_status = ENGINE_PREDIP_MODE;

        predip_timer_1 = 0;
        predip_timer_2 = 0;
        predip_timer_3 = 0;

        if ((actual_engine_pct_trq < 5) || (forced_predip_timer > 0))
            predip_timer_1 = PREDIP_NORMAL_TIME;
        else
            desired_engine_pct_trq = actual_engine_pct_trq;
    }

    engine_control = TORQUE_CONTROL;
    command_ETC1 = C_ETC1_OVERSPEED;

    if ((intent_to_shift_switch == TRUE)) /* Allows for recovery mode if the "intent" switch */
        positive_pedal_trans = TRUE;           /* is released before neutral is achieved */

    if (predip_timer_1 < PREDIP_NORMAL_TIME)
    {
        if ((desired_engine_pct_trq >= TORQUE_RAMP_OFF_RATE) &&
            (actual_engine_pct_trq > 0))
        {
            desired_engine_pct_trq -= TORQUE_RAMP_OFF_RATE;

            if ((intent_to_shift_switch == TRUE) && /* faster rate for intent to shift */
                (shift_init_type == MANUAL) &&
                (actual_engine_pct_trq > 0))
                desired_engine_pct_trq -= 1;
        }
        else
        {
            desired_engine_pct_trq = 0;

            /* check to force bump if neutral not achieved */
            if (actual_engine_pct_trq < 10)
            {
                if (++predip_timer_3 >= PREDIP_ZERO_FDBK_TIME)
                    predip_timer_1 = PREDIP_NORMAL_TIME;
            }
        }
        ++predip_timer_1;
    }
    else
    {
        if (((lpf_output_accel > -150) || (forced_predip_timer > 0)) &&
            (predip_timer_1 < (PREDIP_NORMAL_TIME + PREDIP_TORQUE_ZERO_TIME)))
        {
            predip_torque_bump_time = PREDIP_TORQ_BUMP_TIME_LO;
            predip_torque_bump_value = PREDIP_TORQ_BUMP_VALUE_LO + needed_percent_for_zero_flywheel_trq;
        }
        else
        {
            if (predip_timer_1 < (PREDIP_NORMAL_TIME + 2*PREDIP_TORQUE_ZERO_TIME))
            {
                predip_torque_bump_time = PREDIP_TORQ_BUMP_TIME_MED;
                predip_torque_bump_value = PREDIP_TORQ_BUMP_VALUE_MED + needed_percent_for_zero_flywheel_trq;
            }
            else
            {

```

```
    predip_torque_bump_time = PREDIP_TORQ_BUMP_TIME_HI;
    predip_torque_bump_value = PREDIP_TORQ_BUMP_VALUE_HI + needed_percent_for_zero_flywheel_trq;
}

if (predip_timer_2 < predip_torque_bump_time)
{
    desired_engine_pct_trq = predip_torque_bump_value;
    if (actual_engine_pct_trq > 0)
    {
        ++predip_timer_1;
        ++predip_timer_2;
    }
    else
    {
        desired_engine_pct_trq = 0;
        ++predip_timer_1;
        ++predip_timer_2;
    }

    if (predip_timer_2 >= PREDIP_TORQUE_ZERO_TIME)
        predip_timer_2 = 0;
}
#endif
```

```

*****
* Function: control_engine_sync (AutoSplit)
*
* Description:
*   This function synchronizes engine speed to output shaft speed
*   during a shift.
*****
static void control_engine_sync(void)
{
    if ((intent_to_shift_switch == TRUE) && /* Intent to shift conditions */
        (shift_init_type == MANUAL))           /* that allow lower offsets. */
    {
        sync_offset_pos = 20;
        sync_offset_neg = -20;
    }

    else
    {
        sync_offset_pos = 45;                  /* normal Autosplit offsets */
        sync_offset_neg = -45;
    }

    if (accelerator_pedal_position > THREE_PERCENT)
        sync_maintain_timer = MAINTAIN_SYNC_TIME;

    if ((engine_status != ENGINE_SYNC_MODE) || (sync_maintain_timer == 0))
    {
        sync_on_timer = 0;
        sync_off_timer = 0;
        sync_first_pass = TRUE;
        sync_first_pass_timer = SYNC_FIRST_PASS_TIME;

        if ((shift_type == UPSHIFT) && (shift_init_type == AUTO))
            sync_offset = sync_offset_neg;
        else
            sync_offset = sync_offset_pos;

        if (engine_status != ENGINE_SYNC_MODE) /* first time through sync */
        {
            engine_status = ENGINE_SYNC_MODE;

            if (forced_prepin == FALSE)
                sync_maintain_timer = MAINTAIN_SYNC_TIME;
        }
        else                               /* sync_maintain_timer reached 0 */
        {
            engine_control = OVERRIDE_DISABLED;
            command_ETC1 = C_ETC1_NORMAL;
        }
    }

    else
    {
        sync_maintain_timer--;

        if (sync_on_timer++ >= 200)      /* allow sync mode for about 2 seconds */
        {
            sync_off_timer = 0;
            engine_control = SPEED_CONTROL;
            command_ETC1 = C_ETC1_OVERSPEED;

            if (sync_first_pass == TRUE)
            {
                if ((shift_type == UPSHIFT) && (shift_init_type == AUTO))
                {
                    sync_speed_modified = (signed int)(input_speed) +
                        (input_speed_accel_filtered / (1000/ENG_RESPONSE_UPSHF_TIME));

                    if (sync_speed_modified < gos_signed)
                    {
                        if (sync_first_pass_timer == 0)
                        {
                            sync_offset = sync_offset_pos;
                            sync_first_pass = FALSE;
                        }
                        else
                    }
                }
            }
        }
    }
}

```



```
*****  
* Function: determine_if_recovery_complete  
*  
* Description:  
*   This routine checks to see if the percent_torque_value_limit has  
*   exceeded the percent_torque_value feedback from the engine by x%  
*   for x milliseconds and will then set percent_torque_value_limit  
*   to 100% to cancel the recovery mode.  
*****/  
  
static void determine_if_recovery_complete(void)  
{  
    if ((net_engine_pct_trq > 10) &&  
        (desired_engine_pct_trq > (net_engine_pct_trq + RECOVERY_CANCEL_OFFSET)))  
    {  
        ++recovery_cancel_timer;  
    }  
    else  
        recovery_cancel_timer = 0;  
  
    if ((recovery_cancel_timer >= RECOVERY_CANCEL_TIME) ||  
        (desired_engine_pct_trq == 100))  
    {  
        /* terminate the recovery mode */  
        desired_engine_pct_trq = 100;  
        engine_status = ENGINE_RECOVERY_MODE_COMPLETE;  
    }  
}  
  
#pragma EJECT
```

```
*****  
* Function: control_engine_recovery_normal  
* Description:  
*   Determine throttle command for recovery mode.  
*   TORQUE_LIMIT is scaled as a BIN 8 number representing the percentage  
*   of torque allowed to the engine during recovery.  
*****/  
  
static void control_engine_recovery_normal(void)  
{  
    engine_control = SPEED_TORQUE_LIMIT;  
    command_ETC1 = C_ETC1_NORMAL;  
  
    desired_engine_speed = 8031; /* torque limit only, max value for speed */  
    torque_limit += RECOVERY_RATE_TABLE[destination_gear+4]; /* BIN 8 */  
    desired_engine_pct_trq = (char)(torque_limit >> 8); /* BIN 0 */  
    determine_if_recovery_complete();  
}  
  
#pragma EJECT
```

```

*****+
* Functions: control_engine_recovery_coasting
*
* Description:
*   Determine throttle command for coasting down shifts mode.
*****
static void control_engine_recovery_coasting(void)
{
    register uint local_uint;

    if (sync_on_timer <= 300)
    {
        ++sync_on_timer;

        engine_control = SPEED_CONTROL;
        command_ETC1 = C_ETC1_NORMAL;

        sync_off_timer = 0;

        /** recov_coast_down_tmp1 = gos + (dgos * K1) - THL0_DS_ENG_DECAY_RAMP **/
        if (dgos < 0)          /* get absolute value */
            _cx = (uint)-dgos;
        else
            _cx = (uint)dgos;

        asm mulu _cxdx, #THL0_DS_ENG_DECAY_K1;      /* BIN 12 */
        asm shr1 _cxdx, #12;                          /* BIN 0 */

        if (_cxdx > 500)                            /* error check */
            local_uint = 0;
        else
            local_uint = _cx;

        if (!lpf_output_accel > 0)
            recov_coast_down_tmp1 = (gos + local_uint) - THL0_DS_ENG_DECAY_RAMP;
        else
            recov_coast_down_tmp1 = (gos - local_uint) - THL0_DS_ENG_DECAY_RAMP;

        /** recov_coast_down_tmp2 = desired_engine_speed - THL0_DS_ENG_DECAY_RAMP **/
        recov_coast_down_tmp2 = desired_engine_speed - THL0_DS_ENG_DECAY_RAMP;

        if (recov_coast_down_tmp1 < recov_coast_down_tmp2)
            desired_engine_speed = recov_coast_down_tmp1;
        else
            desired_engine_speed = recov_coast_down_tmp2;
    }
    else
    {
        if (sync_off_timer <= 5)
        {
            ++sync_off_timer;
            engine_control = TORQUE_CONTROL;
            command_ETC1 = C_ETC1_NORMAL;
            desired_engine_pct_trq = 0;
        }
        else
            sync_on_timer = 0;
    }

    if ((desired_engine_speed + THL0_DS_FINISHED_DELTA) < gos)
    {
        /* terminate the recovery mode */
        desired_engine_pct_trq = 100;
        engine_status = ENGINE_RECOVERY_MODE_COMPLETE;
    }
}

#pragma EJECT

```

```

*****  

* Function: control_engine_recovery  

*  

* Description:  

*   This function determines which type of throttle recovery should be  

*   used. And initializes some of the variables that will be used.  

*****/  

  

static void control_engine_recovery(void)  

{  

    if ((engine_status != ENGINE_RECOVERY_MODE) &&  

        (engine_status != ENGINE_RECOVERY_MODE_COMPLETE))  

    {  

        engine_status = ENGINE_RECOVERY_MODE;  

        desired_engine_pct_trq = 0;  

        recovery_cancel_timer = 0;  

        sync_on_timer = 0;  

        sync_off_timer = 0;  

  

        /* reset pedal transition variables */  

        positive_pedal_trans = FALSE;  

        positive_pedal_trans = FALSE;  

        zero_flywheel_trq_timer = 0;  

        zero_flywheel_trq_time = 0;  

  

        if (gos < desired_engine_speed)  

            desired_engine_speed = gos;  

  

        /* set initial starting torque limit */      /* percent, BIN 8 */  

        if ((actual_engine_pct_trq > needed_percent_for_zero_flywheel_trq) &&  

            (pct_demand_at_cur_sp > 5))  

            torque_limit = ((unsigned int)(actual_engine_pct_trq))<<8; /* percent, BIN 8 */  

        else  

            torque_limit = ((unsigned int)(needed_percent_for_zero_flywheel_trq))<<8; /* percent, BIN 8 */  

    }  

  

    if ((destination_gear > ACTIVE_RECOVERY_GEAR) &&  

        (pct_demand_at_cur_sp < 5) &&  

        ((shift_type == COAST_DOWN_SHIFT) ||  

         (shift_type == UPSHIFT)))  

    {  

        control_engine_recovery_coasting();  

    }  

    else  

    {  

        control_engine_recovery_normal();  

    }  

}  

  

#pragma EJECT

```

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- BLACK BORDERS**
- IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- FADED TEXT OR DRAWING**
- BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- SKEWED/SLANTED IMAGES**
- COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- GRAY SCALE DOCUMENTS**
- LINES OR MARKS ON ORIGINAL DOCUMENT**
- REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.